2004-09

Improving banch-and-price algorithms and applying them to Stochastic programs

Silva, Eduardo Ferreira

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

IMPROVING BRANCH-AND-PRICE ALGORITHMS AND APPLYING THEM TO STOCHASTIC PROGRAMS

by

Eduardo Ferreira Silva

September 2004

Dissertation Supervisor: R. Kevin Wood

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2004	**3. REPORT TYPE AND DATES COVERED** Dissertation	
4. TITLE AND SUBTITLE: Improving Branch-and-Price Algorithms and Applying Them to Stochastic Programs			**5. FUNDING NUMBERS**
6. AUTHOR(S) Silva, Eduardo F.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approval for public release pending; distribution is limited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

The first phase of this research demonstrates improvements in the performance of branch-and-price algorithms (B&P) for solving integer programs by (i) stabilizing dual variables during column generation, (ii) performing strong branching, (iii) inserting multiple near-optimal columns from each subproblem, (iv) heuristically improving feasible integer solutions, and by applying several other techniques. Computational testing on generalized-assignment problems shows that solution times decrease over "naïve" B&P by as much as 96%; and, some problems that could not be solved by standard branch and bound on the standard model formulation have become easy.

In the second phase, this research shows how to solve a class of difficult, stochastic mixed-integer programs using B&P. A new, column-oriented formulation of a stochastic facility-location problem (SFLP), using a scenario representation of uncertainty, provides a vehicle for demonstrating this method's viability. Computational results show that B&P can be orders of magnitude faster than solving the original problem by branch and bound, and this can be true even for single-scenario problems; i.e., for deterministic problems. B&P also solves SFLP exactly when random parameters are modeled through certain continuous probability distributions. In practice, these problems solve more quickly than comparable scenario-based problems.

14. SUBJECT TERMS branch-and-price, integer programming, stochastic programming, generalized assignment problem, facility-location problem.			**15. NUMBER OF PAGES** 99
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

IMPROVING BRANCH-AND-PRICE ALGORITHMS AND APPLYING THEM TO STOCHASTIC PROGRAMS

Eduardo F. Silva
Lieutenant Commander, Brazilian Navy
B.S., Brazilian Naval Academy, 1986
B.S., "Almirante" Wandenkolk Instruction Center, 1990
M.S.I.E. "Fluminense" Federal University-Brazil, 1998
M.S.C.S. "Fluminense" Federal University-Brazil, 2000

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 2004

Author: _____
Eduardo F. Silva

Approved by:

_____	_____
R. Kevin Wood Professor of Operations Research Dissertation Supervisor	Gerald G. Brown Distinguished Professor of Operations Research
_____	_____
Matthew Carlyle Associate Professor of Operations Research	Robert A. Koyak Associate Professor of Operations Research
_____	
Craig W. Rasmussen Associate Professor of Applied Mathematics	

Approved by: _____
James N. Eagle, Chair, Department of Operations Research

Approved by: _____
Julie Filizetti, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The first phase of this research demonstrates improvements in the performance of branch-and-price algorithms (B&P) for solving integer programs by (*i*) stabilizing dual variables during column generation, (*ii*) performing strong branching, (*iii*) inserting multiple near-optimal columns from each subproblem, (*iv*) heuristically improving feasible integer solutions, and by applying several other techniques. Computational testing on generalized-assignment problems shows that solution times decrease over "naïve" B&P by as much as 96%; and, some problems that could not be solved by standard branch and bound on the standard model formulation have become easy.

In the second phase, this research shows how to solve a class of difficult, stochastic mixed-integer programs using B&P. A new, column-oriented formulation of a stochastic facility-location problem (SFLP), using a scenario representation of uncertainty, provides a vehicle for demonstrating this method's viability. Computational results show that B&P can be orders of magnitude faster than solving the original problem by branch and bound, and this can be true even for single-scenario problems; i.e., for deterministic problems. B&P also solves SFLP exactly when random parameters are modeled through certain continuous probability distributions. In practice, these problems solve more quickly than comparable scenario-based problems, with say, 50 scenarios.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Kevin Wood, whose expertise, talent and insight were instrumental in making this journey successful. I am especially grateful for his responsiveness and attention during his sabbatical at the University of Auckland, New Zealand. It is hard to believe it would be possible to receive more attention. I do not remember any e-mail that took more than one day to be replied to, and always with extreme courtesy and patience. I also thank Professor Matt Carlyle for his help and constructive comments.

I would like to acknowledge the Naval Postgraduate School, especially the Operations Research Department and my committee, for their support and hard work, which created the conditions for this degree to become reality.

I am also grateful to the team behind the Computational Infrastructure for Operations Research (COIN-OR), whose contributions made this research possible.

I wish to thank the other PhD students, particularly Keith Olson and Andy Hernandez, for all their help, support, and most importantly, their friendship during my stay in Monterey-CA.

My special thanks to my lovely wife, Rosangela, and to my son, Gabriel, for their understanding and for giving me the necessary balance to guarantee my sanity at the end of this work. Thanks to my family in Brazil for their continuing encouragement as they met many challenges during my absence.

I thank the Brazilian Navy for giving me this exciting opportunity and for supporting me during this time.

Finally, I thank God for showing me some light in hard moments.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Integer programs are used to model real-world decision-making problems involving discrete decisions such as: (*a*) which subset of a set of orders should be assigned to each truck in a fleet of trucks so as to minimize total delivery costs?, (*b*) how should aircrews be assigned to flights to ensure that an airline's schedule of flights is properly crewed at minimum cost?, and (*c*) which set of potential production facilities should be opened to meet future, estimated customer demands at minimum cost? The purpose of this thesis is to provide new methods to solve certain integer programs more efficiently, and to extend those methods to handle uncertainty in model parameters. We are concerned with "column-oriented formulations" of integer programs solved through a "branch-and-price algorithm."

A "column-oriented formulation" of an integer program may, in theory, require much less (branch-and-bound) enumeration to solve than a standard formulation, making it an attractive alternative. On the other hand, it must normally be solved using "dynamic column generation" because typical column-oriented formulations have too many variables (columns) to write down explicitly: The model's variables are generated "on the fly," according to standard optimization theory. Dynamic column generation can directly solve linear programs, but must be embedded within a branch-and-bound algorithm to solve an integer program. The combined technique is called "branch and price." Branch and price is clearly more difficult to implement that standard branch and bound, and, consequently, the body of research on this topic is limited. Room for computational improvements exists, as well as room for a broader class of applications.

The first phase of this research demonstrates improvements in the performance of branch-and-price algorithms (B&P) for solving integer programs by (*i*) stabilizing dual variables during column-generation, (*ii*) performing strong branching, (*iii*) inserting multiple near-optimal columns from each subproblem, and through other techniques. "Duals stabilization" uses standard non-linear programming techniques to improve the column-generation phase of the overall B&P algorithm. And, strong branching is, essentially, a "look-ahead procedure" that decides which fractional variable to branch on by actually performing the branching on each of a set of candidates, and following the

branch that appears most attractive. (This contrasts with standard methods that compute "attractiveness" based solely on information from the current solution.) "Inserting multiple near-optimal columns" simply means that, instead of generating just a few new columns (variables) in each iteration of the algorithm, we generate many.

Computational testing on generalized-assignment problems shows that the computational enhancements described decrease solution times by as much as 96% over "naïve" B&P. Furthermore, some problems that could not be solved by branch and bound on the standard model formulation become easy.

The second phase of this research shows how to formulate and solve a class of difficult, stochastic mixed-integer programs using B&P. A "stochastic program" directly models the uncertainty that one faces in many decision-making problems. For instance, in the facility-location problem alluded to in item (*c*) above, a stochastic program explicitly models the uncertainty in costs, and/or demands and/or capacities. We first show that a number of stochastic programs have column-oriented formulations that are amenable to solution by B&P. We then use a stochastic facility-location problem (SFLP), having a scenario representation of uncertainty, to demonstrate the potential usefulness of our methods. Computational results show that B&P can be orders of magnitude faster than solving the original problem by branch and bound, and this can be true even for single-scenario problems, i.e., for deterministic problems. B&P also solves SFLP exactly when random parameters are modeled through certain continuous probability. In practice, these problems solve more quickly than comparable scenario-based problems, with say, 50 scenarios.

I. INTRODUCTION

Gilmore and Gomory (1961) develop *dynamic column generation* to solve linear programs, or approximations to integer programs, that have too many variables to be explicitly inserted into a mathematical model stored in a computer. The limited computer memories of that era spurred the interest in this technique. Dynamic column generation initially defines a small subset of the full problem's columns (variables), and solves to optimality the resulting restricted linear program (LP). Next, the dual solution for this restricted LP helps generate (identify and create) one or more columns left out of the initial subset that have favorable reduced costs. These columns are inserted into the restricted model, the new restricted LP is solved, and the process repeats until no column with favorable reduced cost can be found. At this point, the optimal solution of the restricted LP is also optimal for the full (linear) problem, and typically, only a small subset of the full model's columns will have ever been generated.

Column generation seemed to lose its attractiveness to researchers and practitioners during the 1970s and 1980s, because computer memory and power improved so much that problems formerly solved with column generation could be solved directly. After all, the solution of a single, explicit linear or integer program is much easier to develop and manage than is an iterative technique like column generation. In the 1990s, however, column-generation reappeared as a key technique to help solve difficult mixed-integer programs (MIPs), e.g., crew scheduling problems (Anbil et al. 1992), and vehicle routing problems (Desrosiers et al. 1995). The key advantage of dynamic column generation in this context is that it can exploit problem formulations that have much tighter linear-programming relaxations than more traditional, compact formulations.

To solve a MIP, column generation must be merged into a branch-and-bound framework, because the column-generation process by itself does not necessarily lead to the generation of all columns needed for a good solution to the MIP. Even a feasible solution is uncertain. Although not the first to use the combined solution technique,

Savelsbergh (1997) labeled the technique "branch and price," and that label now prevails in the literature.

The purpose of this dissertation is twofold: (*i*) to improve the efficiency of branch-and-price algorithms, and (*ii*) to extend the application of branch and price to a class of two-stage stochastic programs. The results of this research are presented in chapters II and III.

Savelsbergh uses the generalized assignment problem for developing the techniques of branch and price. Chapter II of this dissertation builds upon his work, using the same class of problems to demonstrate a number of new techniques that substantially improve the performance of branch and price. These techniques include (*i*) stabilizing dual variables during column generation, (*ii*) performing strong branching, (*iii*) inserting multiple near-optimal columns from each subproblem, (*iv*) heuristically improving feasible integer solutions, (*v*) eliminating "unprofitable columns," and (*vi*) solving only one subproblem at a time, instead of all possible subproblems, before re-solving the restricted master problem.

Having a good implementation of a B&P in hand, chapter III extends its application to a class of stochastic mixed-integer programs (SMIPs). This application of B&P to SMIPs is only the third in the literature, and uses a totally different approach than the first two (Damodaran and Wilhelm 2004, Lulli and Sen 2004; however see also Shiina and Birge 2004 and Singh et al. 2004, who investigate column generation to solve SMIPs, without using a formal B&P algorithm). We demonstrate the B&P approach in detail on a stochastic facility-location problem, where uncertainty is represented by a finite number of scenarios, or through continuously distributed random parameters that satisfy certain assumptions. These models are solved exactly, an uncommon result in the stochastic-programming literature, where probabilistic guarantees for solution quality are the norm (e.g., Laporte and Louveaux 1993, Carøe and Tind 1998, Sen and Higle 2000).

Chapter IV summarizes the contributions of this research recommends areas for further work.

II. IMPROVING BRANCH AND PRICE

A. INTRODUCTION

The seminal work of Ford and Fulkerson (1958), which suggests dealing implicitly with the variables of a multicommodity-flow problem, signals a way to solve linear programs (LPs) with too many variables to list explicitly. We shall call the technique they suggest *dynamic column generation*. Dantzig-Wolfe decomposition (Dantzig and Wolfe 1960) extends this approach to general linear programs. Gilmore and Gomory (1961) then apply this strategy to solve an integer program (IP), specifically, the cutting-stock problem. Gilmore and Gomory round their LP solution to an integer one with good results, but with no method for guaranteeing an epsilon-optimal solution. Appelgren (1969, 1971) applies Dantzig-Wolfe decomposition to a *ship scheduling problem*, an IP, to make it eligible for dynamic column-generation. However, his solution approach cannot guarantee optimal solutions, either. Johnson (1989) proposes embedding dynamic column generation within an integer-programming, branch-and-bound algorithm to overcome this difficulty. Savelsbergh (1997) introduces the now-standard phrase *branch and price* (B&P) to describe this combined technique; however, Desrochers and Soumis (1989), Anbil et al. (1992), and others, use the technique earlier for such applications as routing and scheduling. Our research demonstrates how to improve the performance of B&P algorithms.

We assume an IP formulation with too many columns (variables) to be written down explicitly: if it were not for the huge number of columns, we would prefer this formulation to a more compact one because it possesses a tighter LP relaxation. We begin our solution procedure for the IP by creating a *restricted master problem* (RMP) that contains an explicit subset of the IP's columns; typically, an ad hoc procedure creates these columns. Then, dynamic column generation (*i*) solves the linear-programming (LP) relaxation of the RMP (LP-RMP) (*ii*) identifies one or more new columns with favorable reduced cost by solving one or more optimizing *subproblems*, (*iii*) adds these new columns to the RMP (enlarging the explicit subset of columns under consideration), and (*iv*) repeats steps (*i*) through (*iii*) until the LP-RMP has been solved to optimality.

Normally, we will not have to generate all possible columns for the master problem; only a small fraction of them will be created. This is the key to the value of dynamic column generation.

The column-generation process uses the dual solution of the LP-RMP together with an optimization subproblem to construct one or more columns that have favorable reduced costs. No such column can be contained in the current RMP so the column, or columns, are added to the column subset, and the LP-RMP re-optimized. Typically, the original IP has a special subproblem structure, and the Dantzig-Wolfe reformulation has multiple convexity constraints, each corresponding to a separate column-generation subproblem. Thus, it is natural to generate one column for each convexity constraint, i.e., the best column in each subset of implicit columns.

Unfortunately, optimally solving the LP-RMP by dynamic column generation does not guarantee that the integer solution to that RMP is optimal to the original IP; even integer feasibility is uncertain. This is true because dynamic column generation, fundamentally a linear-programming procedure, may not generate all columns necessary for an IP solution. Finding the optimal IP solution for the full problem requires that some type of branch-and-bound procedure be applied along with further column-generation within the branch-and-bound search tree. This is *branch and price*. (Solution procedures based solely on cutting-plane approaches seem impractical.) Fathoming of nodes in the search tree occurs in the standard fashion: An optimal integer solution to the RMP is identified, or bounding information indicates that any solution satisfying the node's restrictions can be no better than an incumbent solution, or the problem is infeasible.

We demonstrate our techniques for improving the efficiency of B&P with the *generalized assignment problem* (Ross and Soland 1975, Cattrysse and Van Wassenhove 1992). Savelsbergh (1997) investigates B&P for solving this problem, and substantially improves solution times compared to the standard, compact formulation solved by branch and bound. Our work may be viewed as an extension of his research. We reduce solution times for branch-and-price by using features such as:

- stabilizing dual variables during the column-generation phase;

- performing strong branching;
- inserting multiple near-optimal columns from each subproblem; and
- heuristically improving feasible integer solutions.

We have not found these enhancements grouped in any other paper on B&P algorithms. Moreover, we have implemented our enhanced B&P algorithm using a framework provided by freely available, open-source code libraries. Together with the fact that we solve the LP-RMPs using a standard, commercial optimizer, we believe that this will make B&P more accessible to the OR community.

The next section defines the generalized assignment problem. Section C discusses the particulars of branch and price and then covers some implementational issues together with the basic algorithm. Section D explores enhancements to the basic algorithm and section E demonstrates their usefulness with computational tests. The final section, section F, presents conclusions.

B. GENERALIZED ASSIGNMENT PROBLEM

The generalized assignment problem (GAP) describes the problem of finding a minimum-cost assignment cost of m capacity-consuming tasks to n capacitated agents, such that (*i*) each task is assigned to exactly one agent, and (*ii*) the agents' capacities are respected. The problem is NP-hard in the strong sense (Martello and Toth 1990, pp. 6-9), and has been studied extensively in the literature; Cattrysse and Van Wassenhove (1992) and Osman (1995) provide good summaries. The standard formulation for the GAP is:

Indices

$i \in I$ agents

$j \hat{I} J$ tasks

Data [units]

c_{ij} cost of assigning task j to agent i [dollars]

w_{ij} amount of agent i's capacity consumed by task j [hours]

d_i capacity of agent i [hours]

Decision variables

x_{ij} 1 if task j is assigned to agent i, and 0 otherwise

Formulation (GAP)

$$\min_{\mathbf{x}} \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in I} x_{ij} = 1 \qquad \forall j \in J \tag{2}$$

$$\sum_{j \in J} w_{ij} x_{ij} \le d_i \qquad \forall i \in I \tag{3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I, \forall j \in J\,. \tag{4}$$

We shall call the above the *compact formulation* of the GAP.

The compact formulation of the GAP challenges LP-based branch-and-bound solvers despite its apparent simplicity (e.g., Guignard and Rosenwein 1989, Cattrysse et al. 1994, Appleget and Wood 2000). The LP relaxation of GAP can be quite poor. One approach to improving the situation is to use a completely different formulation, a *column-oriented formulation*, which normally has a tighter relaxation. The column-oriented formulation can be derived formally through an extension of Dantzig-Wolfe decomposition (Dantzig and Wolfe 1960, Desrosiers et al. 1995, Wolsey 1998, section 11.2).

We state the GAP's column-oriented formulation next, and refer to it henceforth as "CGAP." CGAP does, indeed, have a tighter LP relaxation than the compact formulation because it eliminates fractional solutions that are not convex combinations of the 0-1 solutions to the knapsack constraints (3).

Indices

$i \in I$ agents

$j \in J$ tasks

$k \in K_i$ joint assignments of tasks to agent i that are feasible with respect to constraint (3)

Data [units]

$\hat{x}_{ij}^{k}$ 1 if task j is assigned to agent i in the k^{th} joint assignment of tasks to agent i

$\hat{c}_i^k$ cost of the k^{th} joint assignment of tasks to agent i ($\hat{c}_i^k = \sum_{j \in J} c_{ij} \hat{x}_{ij}^k$) [dollars]

Decision variables

l_i^k 1 if the k^{th} joint assignment of tasks to agent i is chosen, and 0 otherwise;

Formulation (CGAP)

$$\min_{?} \sum_{i \in I} \sum_{k \in K_i} \hat{c}_i^k l_i^k \tag{5}$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{k \in K_i} \hat{x}_{ij}^k l_i^k = 1 \qquad \forall j \in J \tag{6}$$

$$\sum_{k \in K_i} l_i^k = 1 \qquad \forall i \in I \tag{7}$$

$$l_i^k \in \{0,1\} \quad \forall i \in I, \forall k \in K_i . \tag{8}$$

Typically, the LP relaxation of a column-oriented formulation like CGAP cannot be solved directly due to the large number of columns. Therefore, each K_i is replaced by a subset in an RMP we denote as "RCGAP." To solve the LP relaxation of RCGAP, we generate additional columns in a *subproblem*, on demand. The subproblem associated with agent i, used to check if a column with negative reduced cost exists, is

$$\min_{\mathbf{x}_i} \sum_{j \in J} (c_{ij} - \hat{a}_j) x_{ij} - \hat{b}_i \tag{9}$$

$$\text{s.t.} \quad \sum_{j \in J} w_{ij} x_{ij} \le d_i \tag{10}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I,\ j \in J , \tag{11}$$

where $\hat{a}_j$ is the dual variable in the relaxed RCGAP associated with constraint (6), and $\hat{b}_i$ is the dual of the convexity constraint (7).

C. BRANCH AND PRICE

Branch and price has been found to be useful for solving certain IPs having column-oriented formulations. However, some fundamental difficulties arise when

applying column-generation techniques for LPs within a branch-and-bound algorithm for an IP. As Savelsbergh (2001) notes:

- Conventional branching on variables may be ineffective because fixing variables can destroy the structure of the pricing subproblem.
- Column generation often converges slowly, and solving LPs to optimality may become computationally prohibitive.

As an example of the first issue, suppose one attempts to apply a standard branching rule on a fractional variable l_i^k in the LP relaxation of RCGAP by fixing this variable's value to zero. The subproblem related to agent *i* will probably have to be solved again, and it is quite likely that the same assignment *k* will be found as a solution. If this happens, it will be necessary to find the second best subproblem solution by enumeration, or by adding a constraint that forbids the first. At depth *p* in the branch and bound tree, it may be necessary to find the p^{th} best subproblem solution. This is definitely a more difficult problem: The simple knapsack structure of the subproblem would be lost by adding constraints, or an enumeration algorithm would be required. (Actually, it turns out that enumerating near-optimal assignments, and hence the p^{th} best assignment, using DP is not difficult with the GAP, and we will take advantage of this fact. However, finding near-optimal solutions for more general column-generation problems can be quite difficult.) Moreover, the subproblems formulated as mixed-integer programs (MIPs) become considerably harder to solve by branch and bound as the dual variables converge to their optimal values (Vanderbeck and Wolsey 1996).

These two fundamental issues explain why standard MIP solvers, such as CPLEX, OSL and XPRESS, are not yet ready to embed B&P methodology, and why we must resort to specialized technology. The options for implementing B&P will be discussed in the next section. B&P algorithms are also highly dependent on the problem being solved when compared to the techniques like *branch and cut*, which standard solvers do implement. A key point that contributes to making B&P more challenging is that, in standard branch and bound (and branch and cut), the IP solution always lies

within the convex hull of the IP's LP-relaxation, while in branch and price, this may not be true until the model is solved.

1. Options to Implement Branch and Price

The issues discussed above make it impossible for MIP solvers at the current state of the art to perform B&P. However, these solvers do typically implement branch and cut. In branch and cut, an IP—the columns of this IP are explicit—is also solved by branch and bound. However, the solver attempts, at each node in the enumeration tree, to improve the local lower bound on the optimal objective value by adding *valid inequalities* ("integer cutting planes," or simply "cuts"). These are simply inequalities that satisfy all feasible solutions to the IP. Unfortunately, this framework is inadequate for handling the generation of new variables at each node of the branch-and-bound tree and to perform branching on multiple variables. Consequently, at this time, researchers are forced to use additional software tools to produce an efficient implementation of the main B&P engine.

Several software packages exist for solving problems by branch and price: COIN/BCP, Maestro, MINTO and Symphony. Maestro is a C++ library being developed by ILOG, Inc. (Charbier 2003), but the product is not available for testing as this research is being conducted. MINTO, the Mixed INTeger Optimizer (Savelsbergh and Nemhauser 1998), is available, free of charge, from the Georgia Institute of Technology. However, it is not open-source and only compiled libraries are provided (http://www.isye.gatech.edu/faculty/Martin_Savelsbergh/software). We have chosen the COIN/BCP library for implementing B&P because all the source code is freely available and it can be linked to different solvers (http://www.coin-or.org). SYMPHONY (SYMPHONY 2004) is another open-source library that presents a branch-and-price framework. But, according to the SYMPHONY web site (http://branchandcut.org/SYMPHONY/faq.htm), "COIN/BCP has improved on SYMPHONY in some areas, however, such as support for branch and price. SYMPHONY's support for branch and price is limited." It also states that the two packages will merge in a near future.

2. COIN/BCP

The COmputational INfrastructure for Operations Research (COIN-OR, or simply COIN) is described in its homepage (http://www.coin-or.org) as "an initiative to spur the development of open-source software for the operations research community." COIN-OR may be viewed as a repository of distinct libraries that can be integrated to build optimization algorithms. The libraries are managed by different projects, such as BCP: a parallel branch-cut-price framework; CGL: a cut-generation library; CLP: COIN native simplex solver; and OSI: an open solver interface layer.

The Branch-Cut-Price (BCP) project (Lougee-Heimer 2003) presents a C++ framework for solving mixed-integer programs, in parallel, on a distributed network of workstations, using LP-based branch-and-cut and/or branch-and-price techniques. IBM research has employed BCP successfully in several commercial projects (Anbil et al. 1999, Ladanyi et al. 2001). BCP is now integrated with the Open Solver Interface, and is in the process of being integrated with the Cut Generation Library (Ralphs and Ladanyi 2001).

A key advantage to BCP is that all the source code is freely available. The IBM Corporation hosts the project by maintaining infrastructure to control code updates as well as maintaining a mailing list for COIN users. Unfortunately, BCP is not formally compatible with the Windows operating systems; this has presented an extra challenge in our work. (Indeed, several changes in BCP software have resulted from our research.) Additional disadvantages include incomplete documentation, and the fact that support is limited to the collaboration coming from a mailing list. The reader should also note that BCP is designed for a parallel/distributed environment, and executing the code sequentially requires a protocol to emulate the parallel environment, and this entails some computational overhead (Ralphs and Ladanyi 2001).

The COIN-OR homepage (http://www.coin-or.org) and Lougee-Heimer (2003) explain why the idea behind this initiative seems promising. Both sources point out that when an optimization algorithm is published, it is likely that other researchers will identify improvements. However, testing new ideas typically requires re-implementing (and re-debugging and re-testing) the original algorithm. Often, clever implementational

details remain unpublished, and it can be difficult to replicate reported performance. If the original algorithm is made available in a community repository, weeks of re-implementing will no longer be required. A researcher can simply check out a copy of the code and make the modifications as necessary. This software reuse may translate into improved productivity. It is expected that papers from researchers using the COIN-OR library will start appearing in the literature (e.g., Lulli and Sen 2004).

3. Basic Algorithm

We describe a generic B&P algorithm next, and follow with certain details. Note that we deal only with minimizing problems, so *lower bound* means an optimistic bound on the optimal objective value and *upper bound* means a pessimistic one. We assume that the algorithm below is solving a MIP that has at least one feasible solution and that the RMP is always feasible. (These assumptions are discussed in section b.)

Basic Branch-and-Price Algorithm.

Input: The data for a compact MIP.

Output: The optimal objective value z^* for the compact MIP and the solution $\mathbf{x}^*$ for its column-oriented version, "CMIP".
/* Conversion of $\mathbf{x}^*$ into the solution for the compact MIP is always straightforward */

Step 1. Initialize an RMP for CMIP with some small set of columns.
Insert the RMP into the branch-and-bound tree as the root node, and mark this node as a candidate node.
Set the root node *local lower bound* $\underline{z} \leftarrow -\infty$.
/* Each node has a local lower bound, $\underline{z}$, associated with it. */
Set the *global upper bound* $\overline{z} \leftarrow +\infty$.

Step 2 Delete all candidate nodes having $\underline{z} > \overline{z}$.
If there are no more candidate nodes then
Print the CMIP's optimal solution value $\overline{z}$ and its optimal solution $\mathbf{x}^*$.
STOP.

Step 3 Select and remove a candidate node from the branch-and-bound tree, according to some user-defined search strategy.
Call the RMP at this node "the current RMP."

Step 4 Optimally solve the LP relaxation of the current RMP for $\hat{\mathbf{x}}$, objective value $\hat{z}$ and dual variables $\hat{\mathbf{p}}$.

Step 5 If $\hat{\mathbf{x}}$ is integral and $\overline{z} > \hat{z}$, then let $\overline{z} \leftarrow \hat{z}$ and $\mathbf{x}^* \leftarrow \hat{\mathbf{x}}$.

Step 6 Use $\hat{\mathbf{p}}$ to generate one or more new columns that price favorably for the column-oriented formulation. /* As in equations (9)-(11), for example */

Step 7 If there exist favorable columns, then
Insert them into the current RMP and go to step 4.
Else If $\overline{z} \leq \hat{z}$ or $\hat{\mathbf{x}}$ is integer, then
Fathom this node and return to step 2.

Step 8 Partition the current RMP into new problems according to the user-defined branching rules.
Add the new problems into the branch-and-bound tree as candidate nodes, and set their *local lower bounds* to $\hat{z}$.
Go to step 2.

a. Initialization

The LP solved at the root node of the branch-and-bound tree requires some initial columns. To adopt a standard procedure for all problems, the initial subset of columns for each agent contains (*i*) the optimal solution of a knapsack problem that maximizes the assignment cost of allocating the tasks for that agent, and (*ii*) a null assignment. It may seem odd to start with maximum-cost columns, but we do not find any differences in solution speeds when we use low-cost columns. We do find that having "some columns" here is useful, however, and maximum-cost ones are easy to compute. The null assignments help the algorithm find feasible solutions early in the solution process, and help simplify the overall procedure, as described below.

b. Infeasibility

After branching, a local LP-RMP may become infeasible. However, this LP-RMP is truly infeasible only if feasibility cannot be recovered by generating new columns. Dual extreme rays can be used to recover feasibility (e.g., Bertsimas and Tsitsiklis 1997, Section 4.8), or we can reformulate the model so that no LP-RMP is ever infeasible. We prefer the latter option for its simplicity, and reformulate the original GAP to allow penalized non-assignment of any task, i.e., to make constraints (2) "elastic." Consequently, constraints (6) in CGAP become elastic, and together with the initial null-assignment columns, this ensures that every LP-RMP has a feasible solution.

c. Updating the Lower Bound

During the column-generation phase, the LP lower bound for CGAP can be continuously updated based on the subproblems' reduced costs (e.g., Lasdon 1970, section 3.7). As the RMP-LP is being solved at a node in the enumeration tree, a lower bound on its LP solution, and hence its IP solution, is $z_{LP-RMP} + \sum_{i \in I} \overline{c}_i$, where $z_{LP\text{-}RMP}$ is the current optimal objective value of the LP-RMP and $\overline{c}_i$ is the minimum reduced cost for columns associated with subproblem *i*.

Continuous updating of the lower bound can reduce computational effort in two ways: (*i*) If that bound exceeds the current upper bound, the node may be fathomed even before local column-generation is complete, and (*ii*) the algorithm can use the information the bound provides to branch before actually solving the local LP-RMP completely. The latter ability can help the algorithm avoid stalling, a behavior that has been reported in set-partitioning problems (Vanderbeck and Wolsey 1996**).**

d. Branching

Standard branching on fractional variables of problems when using dynamic column generation may impose some difficulties, as discussed at the beginning of this section. To overcome this, alternative branching rules can be used based on compact model's variables. The values of those variables can be easily calculated on the

fly from the current l_i^k variables, and used to drive the branching process. We call this *implicit-constraint branching* (Ryan and Foster 1981, Appleget and Wood 2000). This strategy, which is standard for B&P algorithms (e.g., Savelsbergh 1997, Barnhart et al. 1998, Barnhart et al. 2000), maintains the subproblem structure and provides a clear way to partition the solution space when branching.

Assume, for instance, we are branching on a fractional variable x_{ij} implied from a solution to RCGAP; two new RMPs are generated. In the first one, x_{ij} is set to 1, meaning that all columns that represent a feasible assignment for the agent *i*, and do not have the task *j* assigned, are eliminated. The knapsack subproblem associated with agent *i* must also be adjusted by fixing x_{ij} to 1, which is straightforward to accomplish. The second RMP has x_{ij} set to zero, so columns that include task *j* assigned to agent *i* are excluded, and x_{ij} is set to 0 in the corresponding knapsack subproblem.

D. POTENTIAL ENHANCEMENTS

This section presents the techniques we have developed to improve the performance of B&P.

1. Stabilizing Dual Variables

First, let us define one iteration of the column-generation phase in the B&P algorithm to consist of one solution of the LP-RMP followed by the identification of one or more new columns to be added to RMP, before it is re-solved (Steps 4 through 6 in the basic algorithm). During the column-generation phase, the dual values of the relaxed RMP do not converge smoothly to the final optimal values of the master problem (Lübbecke and Desrosiers 2002). An effect called *heading-in* arises in early iterations, where, due to the poor dual information, many irrelevant columns are generated (Vanderbeck 2003). This can cause much unnecessary work.

From the dual point of view, adding columns to the LP-RMP is equivalent to adding rows (constraints) to its dual. We are, in fact, maximizing a concave function (the dual solution to LP-RMP) using Kelley's cutting-plane algorithm (Kelley 1960). This

function is not well approximated in early iterations, implying poor initial solutions. In essence, we are completely optimizing a poor approximation of the true function, and this leads to unnecessary work.

A second effect tackled by duals stabilization, called *tailing-off*, is better known because of its presence when generating cuts in branch-and-cut algorithms (Johnson et al. 2000). Tailing-off refers to the large number of column-generation iterations often needed to prove LP optimality (Vanderbeck and Wolsey 1996). One cause of this in our model may be the degeneracy common in set partitioning models (Butchers et al. 2001). Our approach to duals stabilization effectively perturbs the partitioning constraints' right-hand sides, which is known to diminish the effects of degeneracy (Charnes 1952).

Initially, we tried a simple approach to stabilize duals by adjusting the dual vector $\hat{\mathbf{p}}$ with an exponential smoothing filter (Neame 1999). Let $\hat{\mathbf{p}}_t$ denote the dual vector obtained at Step 4 in the algorithm in the t^{th} iteration of the column-generation phase, and let $\hat{\mathbf{S}}_t$ denote the current vector of smoothed duals. The smoothing scheme defines the parameter α, $0 < \alpha < 1$, sets $\hat{\mathbf{S}}_1 = \hat{\mathbf{p}}_1$ and $\hat{\mathbf{S}}_t = \alpha \hat{\mathbf{p}}_t + (1-\alpha)\hat{\mathbf{S}}_{t-1}$ for $t > 1$. This is the *exponential smoothing* formula, and the parameter α is called the *smoothing constant*. The intuition is simple: Especially in early iterations, the approximation of the dual objective function gets worse and worse the farther the new duals move from the current solution; so, it's OK to find a good direction to move, but do not move too far.

Unfortunately, this procedure provides only modest improvements in computational speed. One of the reasons for this poor performance—this contrasts with the substantial performance gains reported by Neame (1999) when solving instances of the cutting-stock problem—might result from the fact that, unlike the cutting-stock problem, more than one subproblem is associated with the RMP for CGAP. Therefore, the duals related to the task-assignment constraints (6) may be adequate for some agents but not for others. However, they are all updated using the same rule. Thus, this simple smoothing method may produce better results when applied to a model having only a single subproblem, e.g., the cutting-stock problem.

To improve control over the dual variables and treat each one more independently, we use a method based on *trust regions* in the dual space. Marsten (1975)

and Marsten et al. (1975) first use such trust regions to improve a column-generation algorithm; they call the technique the *boxstep method*. They solve the Lagrangian dual problem over a box centered on the current values of the dual variables, preventing the algorithm from taking large steps. Their box size is static, i.e., it does not change during the solution process. Du Merle et al. (1999) exploit *dynamic elastic trust regions* for column generation, which we describe next.

Consider first a bounded linear program *P* and its dual *D*:

$$\begin{array}{l|l} (P)\ \min \mathbf{cx} & (D)\ \max \mathbf{bp} \\ \quad \text{s.t. } A\mathbf{x} = \mathbf{b} & \quad \text{s.t. } A'\mathbf{p} \le \mathbf{c} \\ \quad \mathbf{x} \ge 0 & \end{array}$$

Now, assume that the cardinality of $\mathbf{x}$ is large as in CGAP, and that (*P*) will be solved by column generation. To apply a trust region for the dual variables, we insert bounded elastic (artificial) variables $\mathbf{y}^+$ and $\mathbf{y}^-$ inserted into the constraints of *P* to create a new primal problem $\tilde{P}$ and its respective dual $\tilde{D}$:

$$\begin{array}{l|l} (\tilde{P})\ \min \mathbf{cx} - \mathbf{d}^-\mathbf{y}^- + \mathbf{d}^+\mathbf{y}^+ \quad [\text{dual vars.}] & (\tilde{D})\ \max \mathbf{bp} - \mathbf{e}^-\mathbf{w}^- - \mathbf{e}^+\mathbf{w}^+ \\ \quad \text{s.t. } A\mathbf{x} - \mathbf{y}^- + \mathbf{y}^+ = \mathbf{b} \quad [\mathbf{p}] & \quad \text{s.t. } A'\mathbf{p} \le \mathbf{c} \\ \quad \mathbf{y}^- \le \mathbf{e}^- \quad [-\mathbf{w}^-] \quad (12) & \quad -\mathbf{p} - \mathbf{w}^- \le -\mathbf{d}^- \quad (13) \\ \quad \mathbf{y}^+ \le \mathbf{e}^+ \quad [-\mathbf{w}^+] & \quad \mathbf{p} - \mathbf{w}^+ \le \mathbf{d}^+ \\ \quad \mathbf{x}, \mathbf{y}^+, \mathbf{y}^- \ge 0 & \quad \mathbf{w}^-, \mathbf{w}^+ \ge 0 \end{array}$$

As noted by Du Merle et al. (1999), if the last two constraints in $\tilde{D}$ are written as $\mathbf{d}^- - \mathbf{w}^- \le \mathbf{p} \le \mathbf{d}^+ + \mathbf{w}^+$, and we set $\mathbf{e}^- = \mathbf{e}^+ = \infty$, we see the standard "hard" trust region of the boxstep method appears: Dual variables must lie in the interval $[\mathbf{d}^-, \mathbf{d}^+]$. When

$\mathbf{e}^-$ and $\mathbf{e}^+$ are finite positive numbers, the dual variables can slip outside that interval, but incur a finite objective-function penalty as they do. Figure 1, from Desrosiers et al. (2002), shows the penalties policy in the dual space, in reference to an arbitrary element of the dual vector $\mathbf{p}$, with $\hat{p}$ representing the center of the trust region.

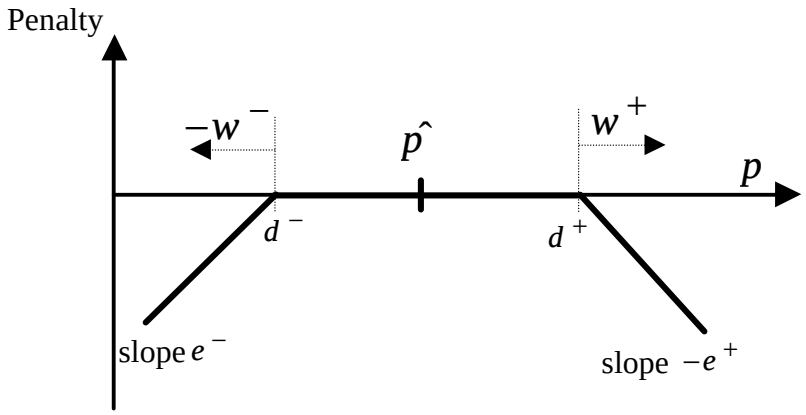

Figure 1. Penalty policy in the dual space.

To update the trust region, we (*i*) re-center the dual value and enlarge the interval, if the new dual solution is outside the interval, or (*ii*) re-center the dual value and reduce the interval, if the new dual is inside the interval. The stopping criteria for the stabilized column generation are that (*i*) all subproblem solutions have non-negative objective values, and (*ii*) $\mathbf{y}^- = \mathbf{y}^+ = \mathbf{0}$.

2. Potential Enhancements in the Column-generation Phase

The subproblems in CGAP are 0-1 knapsack problems, which are IPs that can be solved by standard LP-based branch and bound or using some specialized methodology (e.g., Martello et al. 2000). We use a standard dynamic-programming algorithm (DP) to solve our knapsack problems (e.g., Martello and Toth 1990, Section 2.6). A key advantage of DP is that the effort required to solve a subproblem does not materially change from iteration to iteration. If solved with branch and bound, the subproblems might become considerably harder to solve as the duals variables of the LP-RMP converge to their optimal values (Vanderbeck and Wolsey 1996).

We also apply some preprocessing to reduce a subproblem's effective size and thereby reduce its solution time. Clearly, any variable x_{ij} in the subproblem (9)-(11) with effective cost $c_{ij} - a_j - ß_i > 0$ will be 0 in an optimal subproblem solution, so all such variables can be eliminated (for the time being). Any variables x_{ij} that are fixed by branching also reduce the effective size of agent *i*'s subproblem.

a. Inserting Multiple Columns for each Subproblem

Instead of only inserting into the RMP a single column from an optimal subproblem solution, an alternative approach also inserts near-optimal solutions, hoping that some good columns can be generated in advance. All columns with a favorable reduced cost are eligible to be inserted into the RMP in any iteration. (Task-to-agent assignments with effective cost $c_{ij} - a_j - ß_i > 0$ cannot be removed from the subproblem if all these columns are to be generated, however.) Unfortunately, the time required to find all such assignments would normally be prohibitive. Moreover, most of those columns will not be used in an optimal solution and their inclusion in the RMP could increase its solution time.

The natural way to deal with this issue is to limit the number of columns being inserted. Trying to balance the trade-off between the time searching and the ability to select the near-optimal solutions, we have implemented an algorithm based on the shortest-path-enumeration algorithm presented by Carlyle and Wood (2003), which identifies near-optimal solutions within a factor $1+?$ of the best solution for any $? = 0$. Surprisingly, the number of near-optimal solutions can still be quite large as the size of the problem grows, even for small *?*. Thus, we simply set *?* to a small value and enumerate near-optimal solutions until some pre-specified limit is reached.

The near-optimal solutions we generate can be biased by the way enumeration is performed, and we use this to our advantage. Suppose we have found an initial optimal solution to the knapsack problem, and are now in the middle of the enumeration algorithm: We are exploring whether or not unassigned items should be added to a partially filled knapsack as we build to a complete, near-optimal solution. If the unassigned items are sorted so that we first explore those that are in the optimal solution, the near-optimal solutions the enumeration algorithm first discovers will be

similar to the optimal one. If the items are sorted in the opposite order, then the first near-optimal solutions found will tend to be dissimilar to the optimal one. We call the latter solutions *nearly-orthogonal* near-optimal solutions. Nearly-orthogonal near-optimal solutions work better in our tests. The motivation to consider nearly-orthogonal assignments is clear: Optimal columns generated by the subproblems tend to cover the same tasks for all agents, but only one agent can have a given task in an optimal solution.

We note that Ryan (2004) uses a different method to achieve the same end. He generates multiple columns for a set-partitioning model during each phase of dynamic column generation and desires that that the rows covered by these columns be "balanced." To do this, he simply tracks how many times a particular row is covered by the columns being generated, and stops generating any new columns for any row whose "coverage limit" has been reached. We have not experimented with such a scheme.

b. Solving Only One Subproblem in each Column-generation Iteration

Decomposition algorithms for mathematical programs often lead to independent, easy-to-solve subproblems. If the master problem is difficult to solve, as it may be with a MIP being solved by Benders decomposition (Benders 1962), then it seems sensible to solve all subproblems in each iteration of the algorithm: Get as much information as possible out of the easy-to-solve subproblems before going back to the difficult master problem. However, if we solve all subproblems in each iteration of B&P, and generate a column from each, those columns will likely overlap in the row coverage they provide, since their objective values are based on the same dual variables. However, the final integer solution can only contain a non-intersecting subset of those columns, so we may be generating many unnecessary columns. Since our master problem is easy to solve, why not generate and add just a single column to the RMP and then re-solve the master problem? Furthermore, when considering just the LP-RMP, the reduced cost of a single column is more representative for the magnitude of a slope in object function improvement than the sum of the favorable reduced costs.

To implement the technique of generating only a single favorable column per iteration, we begin by randomly ordering the agents. Then, in that order, we begin

solving the subproblems and stop with the first subproblem that yields a favorable column. That column inserted into the LP-RMP; the LP-RMP is re-solved; and its new dual variables extracted. We then re-randomize the order of the subproblems and start the next iteration. (Other randomization schemes are possible, of course, but this one is easy to implement and works well.)

3. Heuristically Improving Integer Feasible Solutions

When an integer feasible solution is found, it is possible to take some advantage of it by searching for better solutions in its neighborhood. Finding superior integer solutions can accelerate B&P algorithms by (*i*) allowing the current node, or even other nodes in the branch-and-bound tree, to be pruned earlier, and by (*ii*) identifying new columns to be inserted to the RMP. These new columns may not have a favorable reduced cost using the current duals because they were generated ahead of the column-generation process. We note that the neighborhood search is not restricted by the active branching rules in the current node. The B&P algorithm accepts any improved solution. However, only columns that satisfy locally active branching rules should be inserted into the RMP of the current node; otherwise, the branch-and-bound scheme will be disrupted.

For the GAP, we run a greedy heuristic each time an integer feasible solution is found. The neighborhood search consists of a "1-opt heuristic" followed by a "2-opt heuristic" (Lin 1965). The 1-opt heuristic attempts to improve the current solution's objective value by reallocating a single task from the currently assigned agent to some other agent with sufficient capacity to add that task to his workload. The 2-opt heuristic tries to improve a solution by swapping two tasks between agents subject to both agents' capacity constraints staying satisfied. These procedures are repeated until no improvements can be found. As one would expect, these heuristics have been used before for solving the GAP (e.g., Osman 1995, Chu and Beasley 1997, Savelsbergh 1997).

4. Other Potential Enhancements

Before presenting additional potential enhancements to B&P, we point out a feature that we consider a must in any algorithm based on solving LPs in a branch-and-

bound tree, which is the use of *warm starts* (or "basis reuse"). Each time the LP-RMP must be resolved, the previous basis is recovered and used to speed the solution of the modified problem. COIN/BCP and COIN/OSI easily handle this feature. Moreover, a warm start is also used when re-solving the restricted master problem after branching.

a. Strong Branching

In order to improve branching efficiency without increasing subproblem complexity, an approach called *strong branching* can be applied (Johnson et al. 2000). Initially, a small set of branching candidates x_{ij} are chosen. For each of these candidates, the LP-RMP will be branched on and have its child nodes solved. The algorithm then chooses the branching variable based on the objective values of the children: We choose the branch having a child with the lowest lower bound. (Essentially, this implements a "one-step look-ahead heuristic.") Basis-saving and warm-start features make this procedure reasonably efficient.

b. Including Variables from the Compact Formulation

Although we are using GAP to demonstrate our B&P enhancements, we wish to cite a feature that is more specific to problems that could be modeled through a column-oriented formulation if it were not for certain side constraints. These additional constraints may destroy the structure of the pricing subproblem and make it difficult to solve (Vanderbeck 2000); branch and price might not appear so attractive in this case. In the case of the GAP, side constraints that connect agents, e.g., priority assignments associated with seniority, would likely make DP solutions of subproblems impossible. But, suppose such constraints, say $\mathbf{x} \in \mathbf{X}'$, are simple to state in the compact formulation. Then, we may simply change the master problem to explicitly incorporate the variables from the compact formulation, by adding the constraints

$$\sum_{k \in \bar{K}_i} \hat{x}_{ij}^k l_i^k - x_{ij} = 0 \quad \forall i \in I,\ j \in J \qquad [q_{ij}] \tag{14}$$

$$x_{ij} \in X' \ \forall i \in I,\ j \in J\ , \tag{15}$$

where q_{ij} represents the dual variable associated with its respective constraint(14).

The objective function of the revised subproblem for agent i becomes $\min \sum_{j \in J} (c_{ij} - \hat{a}_j - \hat{q}_{ij}) x_{ij} - \hat{b}_i$, where all data and variables are the same as presented before. As one can see, the additional constraints (14) together with the side constraints (15) do not change the subproblem structure.

c. Deleting Poor Candidate Columns

This feature deletes, at every specified number of column-generation iterations, all columns with reduced cost above a given threshold. Intuitively, these columns are unlikely to contribute to the optimal solution, or at least to the optimal solution in the current branch of the enumeration tree. These columns increase the size of the RMP and, consequently, increase solution times. Many of these "poor columns" are generated at the beginning of the B&P algorithm and, the first few times this deletion procedure is executed, quite a few are eliminated. In the final steps of the algorithm, this feature has little effect. The enhancement has a parallel in the dual space, where non-binding constraints can be deleted in Benders decomposition, if done judiciously (e.g., Alvarez 2004).

E. COMPUTATIONAL RESULTS

BCP/COIN is originally designed to be executed in a parallel/distributed environment, and the protocol that emulates this environment in a serial one incurs some computational overhead. We prefer to report results that exclude this overhead, because it would not be difficult to eliminate it in a proper serial implementation. Therefore, we report four different measures in the computational tests. These measures are the total time solving the RMP relaxations (TSLP), total time solving the subproblems (TSS), total time spent branching (TB) and the total time (TT), which is the sum of the first three. TSS includes the time required to find and insert new columns, and TB includes the computation time spent performing strong branching, when that option is activated. We note that, for our implementation, the overhead for a small problem seldom exceeds 50% of overall running time and its maximum value is 0.56 CPU seconds. For problems with larger running times, overhead rarely exceeds 10% of the overall running time.

Our algorithm is implemented using COIN's open solver interface (OSI), coupled with CPLEX 8.0. Tests are carried out on a networked workstation with a 2 GHz Pentium 4 processor and 1 GB of RAM. All problems are solved to optimality and the times are measured in CPU seconds. We have also implemented a procedure to submit the GAP problems, in their compact formulation, to an IP solver. These binary IPs are solved by CPLEX 8.0 using default options and solution times are presented under the heading "IP CPLEX." (We did not find any changes from the default options that produce consistently better solution times for these GAPs.)

1. Parameter Settings and Other Details

Many parameters that affect the performance of the computational enhancements must be set for testing. We choose the parameter values empirically, after an initial set of testing considering problems of various sizes. Therefore, the values we use might not be the best settings for a specific problem instance.

a. Duals Stabilization

In the duals stabilization, the initial interval for the duals, $[\mathbf{d}^-, \mathbf{d}^+]$, is set to [- 50, - 10]. The dual penalties $\mathbf{e}^-$ and $\mathbf{e}^+$ are randomly chosen in the interval [15,20]. The other additional parameter we use is just a scale factor defining how much the trust region and the penalties will be enlarged and shrunk. We use 0.07 for this value in all tests, meaning, for example, that the dual interval is multiplied by 1.07 for enlargement, and by 0.93 for shrinking. We decrease the dual penalties until they cross a threshold value, set to 10^{-5}. After that, we keep their small values to help avoiding degeneracy, as explained previously.

b. Strong Branching

When using strong branching, the number of candidate variables selected for branching is two. Strong branching on a standard (compact) integer program would use a larger number, i.e., check more branches, but the amount of work required to carry out strong branching there is much less than when using column generation.

c. Deleting Variables with Strongly Unfavorable Reduced Cost

Columns with strongly unfavorable reduced cost are deleted after every 50 column-generation iterations. A column is deleted if its reduced cost is greater than 5% of the current objective value of the RMP.

d. Solving One Subproblem at each Column-generation Iteration

There are no parameters for this feature.

e. Heuristic Improvements in Integer Feasible Solutions

There are no parameters for this feature.

f. Inserting Multiple, Near-orthogonal Columns

The maximum number of near-optimal columns inserted is 10, and a column is deemed near-optimal only if it is within 20% of being optimal.

2. Initial Tests: Basic B&P and B&P with Duals Stabilization

We use three sets of GAP instances to evaluate our proposed enhancements to B&P. The first two sets come from the OR-Library (Beasley 1990), while the third set represents instances of "class D" as described by Guignard and Rosenwein 1989 and Romeijn and Morales 2001. The class-D problems have correlations between the c_{ij} and w_{ij}, which make them harder to solve.

The first set of problems originates from the OR-Library (Beasley 1990). The full set consists of twelve groups, each with a distinct combination of number of agents and number of tasks. Each group contains five problem instances. For testing purposes, we select a subset consisting of eight problem groups, shown in Table 1. (We have omitted a few of the smaller problem groups because they solve too quickly and do not add any insight.)

We refer to these as "regular problems," because they appear in several studies in the literature (e.g, Amini and Racer 1994, Osman 1995, Chu and Beasley 1997, Savelsgergh 1997). Although these problems are available in the OR Library, their compact formulations do not represent challenges for a state-of-the-art solver such as CPLEX, which can solve any of them in less than 1 CPU second on our computer.

We have a goal in exploring the regular problems, however: We want to show that B&P does not spend excessive time trying to solve them compared to a standard, LP-based branch-and-bound code. We also show solution times for the algorithm described by Karabakal et al. (1992). (The original code was obtained from the authors.) This branch-and-bound algorithm incorporates an improved multiplier adjusting method (Fisher et al. 1986, Guignard and Rosenwein 1989), which is used to solve a Lagrangian relaxation of the GAP and derive improving feasible solutions. The table below presents the results for each selected group, where each row shows average times across all five problems.

Problem Group	Agents	Tasks	IP (CPLEX)	Karabakal et al. (1992) algorithm	Basic B&P algorithm	Enhanced B&P Algorithm: Duals stabilization
gap1	5	15	0.02	0.00	0.02	0.02
gap4	5	30	0.08	0.13	0.26	0.24
gap5	8	24	0.07	0.07	0.07	0.06
gap8	8	48	0.61	9.35	1.04	0.80
gap9	10	30	0.10	0.27	0.09	0.11
gap10	10	40	0.21	4.30	0.26	0.22
gap11	10	50	0.18	4.54	0.50	0.48
gap12	10	60	0.26	46.59	1.89	1.18

Table 1. Average solution times (CPU seconds) for small generalized assignment problems we label as "regular problems." Each time represents the average over five instances.

The computational times in Table 1 are too small to get the right idea about the impact of the enhancements we present in this research. However, one can see that the differences between IP and the enhanced B&P are modest, indicating that B&P is a

reasonable choice for these problems. This contrasts with the idea that B&P is only suitable for huge problems (Barnhart et al. 1998). The performance of Karabakal's algorithm seems to suffer considerably when problem sizes increase. It performs better than IP (CPLEX) only for the smallest problems, and then only by a small amount. Therefore, to validate the benefits of the proposed enhancements we use harder problems and compare the results with the IP (CPLEX) performance. We will, however, utilize the Karabakal et al. algorithm for certain other comparisons because Savelsbergh (1997) compares his algorithm to it.

Solutions for a second set of problem instances demonstrate that, when regular problems are "scaled up," the basic B&P solution times can be significantly improved with a few enhancements. The second set of test problems also originates from the OR Library (Beasley 1990); Chu and Beasley (1997) categorize these as "large-sized problems" in their tests with genetic algorithms. Table 2 shows problem statistics and how we have named these problems. They are organized classes in according the way they are randomly generated (Martello and Toth 1990, Section 7.6): Problems of class A generally admit many feasible solutions. Problems of classes B and C have tighter capacity constraints, and, consequently, fewer feasible solutions. Table 2 also presents the optimality gap for the compact and column-oriented formulation, so it is possible to see that the column-oriented formulation is substantially tighter.

Problem	Agents	Tasks	Compact formulation: optimality gap (%)	Column-oriented formulation: optimality gap (%)
A-1	5	100	0.02	0.00
A-2	10	100	0.11	0.00
A-3	20	100	0.08	0.00
B-1	5	100	0.64	0.23
B-2	10	100	0.45	0.00
B-3	20	100	0.94	0.00
C-1	5	100	0.36	0.07
C-2	10	100	1.08	0.15
C-3	20	100	1.97	0.11

Table 2. Optimality gaps for large-sized test problems from Chu and Beasley (1997).

The first enhancement tested is duals stabilization. Table 3 shows that stabilization decreases the time spent in the column-generation phase, positively affecting TSS and TB. (IP times are also presented for comparison.) The next table presents another statistic called *reduction factor* (R_F), defined as

$$R_F = 1 - \frac{\text{TT after additional enhancements has been applied}}{\text{TT for the algorithm in its previous configuration}}.$$

Problem	IP CPLEX		Basic B&P algorithm: No enhancements						Enhanced B&P algorithm: Duals stabilization						R_F
	NN	Time	NN	COLS	TSLP	TSS	TB	TT	NN	COLS	TSLP	TSS	TB	TT	
	(nodes)	(sec.)	(nodes)		(sec.)	(sec.)	(sec.)	(sec.)	(nodes)		(sec.)	(sec.)	(sec.)	(sec.)	
A-1	1	0.02	1	5206	86.94	1.49	0.00	88.43	1	1893	29.68	0.53	0.00	30.20	0.66
A-2	1	0.05	3	2901	9.11	0.36	0.06	9.53	5	1608	8.33	0.19	0.06	8.58	0.10
A-3	1	0.05	1	1875	2.83	0.22	0.00	3.05	5	1342	2.61	0.13	0.03	2.77	0.09
B-1	825	1.86	47	11131	121.76	5.33	7.36	134.46	57	5952	60.89	3.13	4.05	68.07	0.49
B-2	1	0.10	1	2142	8.89	0.20	0.00	9.09	1	1443	6.66	0.28	0.00	6.94	0.24
B-3	1	0.10	5	1559	3.30	0.06	0.06	3.42	5	1239	2.11	0.06	0.03	2.20	0.36
C-1	147	0.32	21	4922	70.52	2.28	2.16	74.95	29	3081	36.22	1.72	1.95	39.89	0.47
C-2	869	3.36	35	3271	17.73	0.55	1.28	19.56	31	2014	10.97	0.38	0.64	11.99	0.39
C-3	464	3.40	13	1997	4.29	0.17	0.30	4.76	15	1877	3.83	0.17	0.11	4.11	0.14

Legend:

NN: Number of nodes in enumeration tree
COLS: Total number of columns generated to solve the problem
TSLP: Time spent solving the linear relaxation of the restricted master problem
TSS: Time spent solving the subproblems
TB: Time spent branching
TT: TSSLP+TSS+TB
R_F: Reduction factor for enhanced B&P over basic B&P.

Table 3. Solution statistics for large-sized test problems from Chu and Beasley (1997). This table compares IP (CPLEX), basic B&P and enhanced B&P using duals stabilization.

It is clear that, by reducing the number of columns that needs to be generated, duals stabilization substantially reduces computational time in the column-generation phase, which can be verified by the decrease in the time spent solving the LP-RMP and the subproblems. The improvements are greater for those problems that have a larger ratio between the number of tasks and the number of agents (*task/agent ratio*).

3. Further Tests: Duals Stabilization with Additional Enhancements

Next, we combine duals stabilization with two more enhancements. We apply strong branching and delete columns in the RMP with unfavorable reduced costs. The number of allowed branching candidates is set to two, meaning that, at each branch, four new RMPs are solved, instead of the standard two. Thus, it might be expected that the TB would be slightly larger. However, this extra time is balanced with reduced TSLP, probably because strong branching helps the algorithm choose better paths through the enumeration tree, thereby reducing that tree's size, and, consequently, the total number of columns that are generated. To delete "poor" columns, we require the algorithm to "clean" the RMP every 50 column-generation iterations, eliminating all columns whose reduced cost is greater than 5% of the current objective-function value.

Problem	IP CPLEX		Enhanced B&P algorithm: Duals stabilization						Enhanced B&P algorithm: Duals stabilization Strong branching Deleting columns						R_F
	NN	Time	NN	COLS	TSLP	TSS	TB	TT	NN	COLS	TSLP	TSS	TB	TT	
	(nodes)	(sec.)	(nodes)		(sec.)	(sec.)	(sec.)	(sec.)	(nodes)		(sec.)	(sec.)	(sec.)	(sec.)	
A-1	1	0.02	1	1893	29.68	0.53	0.00	30.20	1	1663	26.24	0.44	0.00	26.68	0.12
A-2	1	0.05	5	1608	8.33	0.19	0.06	8.58	5	1451	7.24	0.25	0.08	7.56	0.12
A-3	1	0.05	5	1342	2.61	0.13	0.03	2.77	1	1237	2.37	0.09	0.00	2.47	0.11
B-1	825	1.86	57	5952	60.89	3.13	4.05	68.07	19	4095	37.22	2.30	2.08	41.60	0.39
B-2	1	0.10	1	1443	6.66	0.28	0.00	6.94	1	1250	6.09	0.17	0.00	6.27	0.10
B-3	1	0.10	5	1239	2.11	0.06	0.03	2.20	7	1207	2.25	0.05	0.06	2.36	-0.07
C-1	147	0.32	29	3081	36.22	1.72	1.95	39.89	11	2291	29.90	1.05	1.09	32.05	0.20
C-2	869	3.36	31	2014	10.97	0.38	0.64	11.99	41	2300	11.81	0.55	1.77	14.13	-0.18
C-3	464	3.40	15	1877	3.83	0.17	0.11	4.11	13	1806	3.36	0.14	0.12	3.62	0.12

Legend:
NN: Number of nodes in enumeration tree
COLS: Total number of columns generated to solve the problem
TSLP: Time spent solving the linear relaxation of the restricted master problem
TSS: Time spent solving the subproblems
TB: Time spent branching
TT: TSSLP+TSS+TB
R_F: Reduction factor for multiply enhanced B&P compare to B&P with duals stabilization only

Table 4. Solution statistics for large-sized test problems from Chu and Beasley (1997). This table compares the effect of strong branching and deleting columns in the RMP, in addition to duals stabilization.

The results in Table 4 show that, even after duals stabilization, additional improvements can be achieved.

To continue testing the proposed enhancements, we now focus on problems whose compact formulations are difficult to solve with a conventional branch-and-bound algorithm. Our goal is to show that B&P can provide solutions to problems that IP simply cannot solve. The next set of problems—they are referred to as "class D" in the literature—exhibits strong correlations between the c_{ij} and w_{ij} (Guignard and Rosenwein 1989). These problems are generated according the following rules (see Savelsbergh 1997, Romeijn and Morales 2001):

w_{ij} is an integer from the discrete uniform distribution on [1,100].

$c_{ij} = 111 - w_{ij} + k$, where k is an integer from the discrete uniform distribution on [–10,10].

$d_i = \frac{0.8}{|I|} \times \sum_{j \in J} w_{ij}$, where $|I|$ is the number of agents.

The next table displays computational results obtained by applying different combinations of the enhancements. The last column describes the enhancements used to solve each problem. The same combination of the enhancements can affect problems of the same size in different ways. Parameter settings are fixed for all tests, independent of problem size. We display results only for the combination of enhancements that provide the best results, as the results vary for different problems.

Agents-tasks	IP CPLEX		Basic B&P Algorithm		Enhanced B&P Algorithm		R_F	Enhancements (see legend)
	NN (nodes)	TT (sec.)	NN (nodes)	TT (sec.)	NN (nodes)	TT (sec.)		
3-30	12729	4.86	795	17.64	91	1.23	0.93	Stz
3-30	30968	11.23	217	4.77	323	4.03	0.15	Stz
3-30	11248	4.69	187	4.67	55	0.67	0.85	Stz
5-30	221693	123.81	2163	19.08	533	5.44	0.71	Stz-SB-D-H
5-30	299936	145.20	2231	21.20	61	0.81	0.96	Stz-SB-D-SO-H
5-30	513014	261.36	3019	28.39	77	1.14	0.96	Stz-SB-D-SO-H
10-30	*	*	827	6.17	171	0.94	0.85	Stz
10-30	*	*	79	0.65	219	1.69	-1.60	Stz-SB
10-30	*	*	295	2.35	267	3.29	-0.40	Stz-SB-SO
5-50	3350	3.47	8679	214.86	581	19.59	0.91	Stz-SB
5-50	128257	116.61	9183	216.38	4859	128.70	0.40	Stz-D-AO-H
5-50	344300	340.73	18075	706.31	7507	346.54	0.51	Stz-SB-SO
10-50	58080	77.89	4933	92.11	1185	21.79	0.76	Stz-D-SO-H
10-50	3390470	6166.37	59929	1096.16	3589	49.39	0.95	Stz
10-50	*	*	15127	340.03	821	22.61	0.93	Stz-SB-SO
5-100	1222683	2705.77	443	665.88	1467	1172.25	-0.76	Stz
5-100	469958	1758.24	*	*	2271	1023.61	>0.85	Stz-SB-SO
5-100	33210	125.34	5177	3434.04	4491	1770.50	0.48	Stz-SB-SO

Legend:
Stz: Duals Stabilization
SB: Strong branching
D: Deleting variables with strongly unfavorable reduced cost
SO: Solving one subproblem at each column-generation iteration
H: Heuristic improvements in integer feasible solutions
AO: Inserting multiple, near-orthogonal columns
* Not solved to optimality after 7,200 CPU seconds.

Table 5. Solution statistics presenting the effect of different combinations of enhancements on test problems of class D.

The results in Table 5 shows that only three easy test problems do not have their solution times improved by enhancements. And, of the fifteen problems that do exhibit improved times, nine have solution times reduced by at least 85%. The CPLEX IP solver is unable to solve the test problems with task/agent ratio of three. However, our enhanced B&P algorithm easily solves those, and even the problems with higher task/agent ratios.

The reduction factor R_F is calculated based on the CPU time. However, if a similar reduction factor based on the number of nodes in the branch-and-bound tree were calculated, it would be almost identical to R_F for problems where $R_F > 0.50$. This means that, for problems where the enhancements provides a significant improvement, most of that improvement results from fewer nodes being explored in the branch-and-bound tree. We conjecture, based on our experience with test problems, that the enhancements, mainly duals stabilization, reduce the number of columns in the RMP, and that having fewer columns somehow makes early branches more effective, allowing faster convergence.

Other potential enhancements were unsuccessful. For instance, forcing early branching when the lower bound does not show improvement for a specified number of column-generations: Here, we detect when the algorithm is stalling by inspecting changes in the object function values. However, there is a clear tendency to branch when the node solution is near the optimum and solution times actually increase. It may be that duals stabilization is already substantially reducing any tailing-off behavior that early branching might usefully address, and thus early branching serves no purpose.

In the next table, we again use the algorithm of Karabakal et al. (1992)—we shall refer to this hereafter as "the K-algorithm"—to compare results for 12 problems types, which also appear in Savelsbergh (1997). Thus, Table 7 summarizes those results and compares some statistics with some from Savelsbergh (1997), who also uses the K-algorithm in his comparisons. We note that this is not a rigorous comparison, because Savelsbergh (1997) uses different hardware and software, and because our test instances are generated the same way, but are not identical. However, as our work is inspired by Savelsbergh's, we think that some comparison should be made.

We choose several tasks-agents combinations, and randomly generate 10 problems instances for each one of the GAP classes A, B and C, following the rules from Savelsbergh (1997). Thus, each row in table 6 contains the average and the maximum over 10 problem instances. The sizes of many of the problems tested by Savelsbergh do not favor our algorithm because it can easily solve them, not leaving much room for observing obvious improvement resulting from our enhancements. For this reason,

among the problem types used by Savelsbergh to compare his algorithm to the K-algorithm, we first choose the problems with 3 and 5 agents, and 50 tasks. The high tasks/agents ratio should stress B&P more than standard branch-and-bound algorithms such as IP CPLEX and the K-algorithm, because such are likely to have many feasible solutions, which also means that the column-oriented formulation is likely to have a high column-to-row ratio. For a second group, we choose the problems with the largest number of variables, which are the ones with 10 and 20 agents, and 50 tasks. The statistics for IP CPLEX are also presented in Table 6.

Problem class	IP CPLEX				Karabakal et al.				B&P + duals stabilization			
Agents-	Nodes		CPU sec.		Nodes		CPU sec.		Nodes		CPU sec.	
tasks	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
A-3-50	1.0	1	0.01	0.05	10.1	61	0.02	0.08	3.4	11	2.69	3.08
A-5-50	1.0	1	0.01	0.02	20.3	98	0.03	0.11	2.6	11	1.20	1.52
B-3-50	75.1	166	0.11	0.20	1304.6	10220	3.74	30.89	18.2	31	3.80	5.82
B-5-50	125.3	510	0.20	0.67	548.9	2055	0.76	3.30	14.2	39	1.76	3.02
C-3-50	83.5	313	0.10	0.28	633.9	2901	1.71	10.20	17.0	35	4.30	7.75
C-5-50	142.6	453	0.23	0.59	367.4	1535	0.52	2.06	8.2	37	1.26	1.91
A-10-50	1.0	1	0.01	0.02	39.0	170	0.07	0.27	1.8	5	0.49	0.56
A-20-50	1.0	1	0.02	0.03	106.5	590	0.22	1.11	2.0	5	0.24	0.30
B-10-50	1.0	1	0.03	0.08	487.6	1395	0.69	2.20	2.2	9	0.37	0.47
B-20-50	1.0	1	0.03	0.05	2735.8	19501	4.33	29.08	3.4	17	0.20	0.35
C-10-50	77.3	248	0.31	0.80	1268.4	3248	1.84	4.64	5.6	13	0.37	0.53
C-20-50	16.6	95	0.23	0.58	8595.7	18918	13.84	27.84	8.4	23	0.20	0.33

Table 6. Solution statistics for problems similar to those in Savelsbergh (1997), comparing the solution times for IP CPLEX, the algorithm of Karabakal et al. (1992) and B&P with duals stabilization. Averages are taken over 10 randomly generated instances. All problems are solved on a networked workstation with a 2 GHz Pentium 4 processor and 1 GB of RAM.

Problems: Class-agents-tasks	Ratio 1: Savelsbergh/ Karabakal et al. (Savelsbergh 1997)	Ratio 2: B&P with Stz/ Karabakal et al. (This research)	Ratio: Savelsbergh/ B&P with Stz (Ratio 1/Ratio 2)	Savelsbergh (1997): % of time spent in column generation	B&P with Stz: % of time spent in column-generation
A-3-50	239.78	134.50	1.78	40	29
A-5-50	21.01	40.00	0.53	50	38
B-3-50	1.64	1.02	1.62	8	5
B-5-50	2.72	2.32	1.17	11	7
C-3-50	10.66	2.51	4.24	7	6
C-5-50	3.87	2.42	1.60	9	12
A-10-50	1.38	6.67	0.21	53	56
A-20-50	0.21	1.09	0.19	63	50
B-10-50	0.63	0.54	1.17	34	45
B-20-50	0.01	0.05	0.32	50	29
C-10-50	0.28	0.20	1.39	16	18
C-20-50	0.02	0.01	1.37	26	12

Legend: Stz: Duals Stabilization

Table 7. Statistics comparing the results of the B&P algorithm presented in Savelsbergh (1997) and the improved B&P with duals stabilizations. Ratios are with respect to solution times. The ratio of ratios in column 4 indicates how much faster or slower B&P with duals stabilization is compared with the Savelsbergh algorithm. Numbers greater than 1 favor our B&P algorithm.

Table 7 only reveals a slight advantage in favor of our enhanced algorithm. However, due to the small solution times involved in this comparison, a more detailed analysis would not be well supported.

F. FINAL REMARKS AND CONCLUSIONS

This research demonstrates improvements in the performance of branch-and-price algorithms (B&P) for solving integer programs by applying enhancements such as stabilizing dual variables during column-generation, performing strong branching, inserting multiple near-optimal columns from each subproblem, heuristically improving feasible integer solutions**.**

We believe that duals stabilization now comprises a necessary feature for any algorithm based on column generation. The graph below (inspired by Kallehauge et al. 2001) shows the effect of applying or not applying duals stabilization (only) for one of

the test problems. The left axis has a logarithmic scale indicating the Euclidean distance between the current dual vector and the optimal one, while solving the linear relaxation of the restricted master problem (LP-RMP) at the first node of a B&P enumeration tree. The thicker line comes from the problem solved with duals stabilization. As the left axis uses a logarithmic scale, the two end-points reflect the last Euclidean distances able to be plotted. Figure 2 also shows the values of the objective function of the LP-RMP on the right axis. This figure shows that stabilization substantially reduces the oscillation of the dual variables (heading-in effect) in the first steps of the column generation and the total number of iterations to find the optimal linear-program (LP) solution for the LP-RMP.

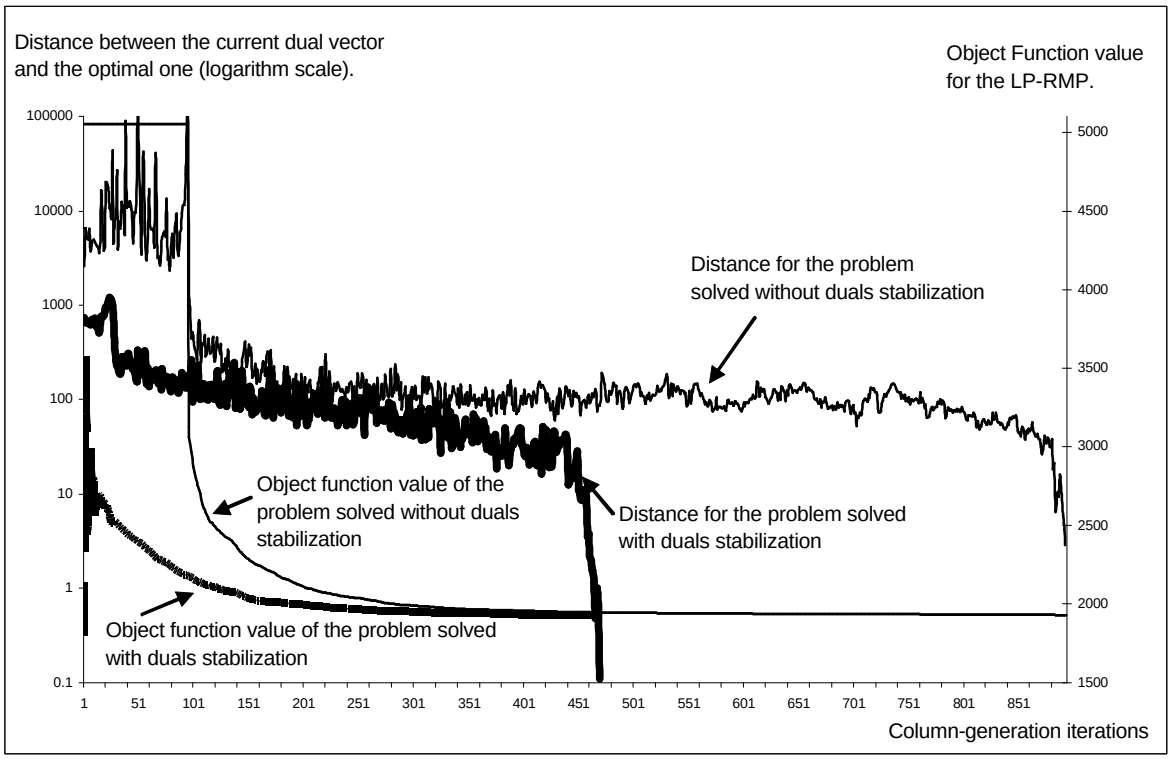

Figure 2. Comparison between the convergence of a LP-RMP being solved with and without duals stabilization for a problem with 8 agents and 48 tasks.

Our computational results show an effect that is opposite to that seen in many studies that solve the compact GAP formulation using branch and bound (e.g., Ross and Soland 1975, Fisher et al. 1986, Guignard and Rosenwein 1989, Marcello and Toth 1990, Karabakal et al. 1992 and Amini and Racer 1994). In all of these references, algorithmic

performance tends to worsen as the task/agent ratio decreases (for a fixed number of tasks). B&P's performance improves as the task/agent ratio decreases. Savelsbergh (1997) notes this also and explains the phenomenon as follows: If the average number of tasks assigned to an agent (task/agent ratio) is small, the knapsack constraints in the subproblems will have only a relatively small number of feasible solutions. This favors column generation because the total number of columns in the explicit column-oriented model will be relatively modest. When the average number of tasks assigned to agents is large, the column-oriented model has many more columns and B&P suffers. The class D test problems in Table 5 also have tighter knapsack constraints and consequently the subproblems in B&P tend to provide fewer feasible solutions than the problems in other classes. That is the reason why our B&P algorithm exhibits outstanding performance with those problems.

The discussion above indicates that B&P will sometimes replace branch and bound as the method of choice to solve an IP. But, one method may be strong while the other is weak, so B&P is best viewed as a complement to branch and bound. However, our research has shown improvements in B&P that should make it a more likely first choice in many cases.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SOLVING A STOCHASTIC FACILITY-LOCATION PROBLEM BY BRANCH AND PRICE

A. INTRODUCTION

This research shows how certain stochastic mixed-integer programs (SMIPs) may be reformulated from their standard forms into column-oriented models, and then explains how to solve such models with a branch-and-price algorithm (B&P). This solution approach is only just starting to be investigated for stochastic programs. However, B&P has proven useful for solving a number of deterministic integer-programming problems (e.g., Savelsbergh 1997, Barnhart et al. 1998, Barnhart et al. 2000), and we shall see that its usefulness can be extended to stochastic programming, too.

We begin by describing a class of mixed-integer, two-stage, stochastic programs whose "normal," compact formulation translates into a column-oriented formulation. We then describe how a number of well-known deterministic models whose stochastic versions fit this framework: the generalized assignment problem (Ross and Soland 1975, Savelsbergh 1997, chapter II of this dissertation), the crew-scheduling problem of Vance et al. (1997), vehicle-routing problems (Desrosiers et al. 1995, Savelsbergh and Sol 1998) and the origin-destination integer multicommodity flow problem (Barnhart et al. 2000). We then describe yet another problem, a stochastic facility-location problem (SFLP), and use it to explore the B&P solution approach in detail.

We initially model and solve a version of SFLP with uncertainty represented through scenarios. Such representations often appear in the stochastic-programming literature (e.g., Butler and Dyer 1999, Chen et al. 2002, Ahmed and Sahinidis 2003, Lulli and Sen 2004). A key advantage to the scenario-based representation is that it allows arbitrary dependence among uncertain parameters. Another common approach in stochastic programming formulates a model through probability distributions, often continuous ones, defined on the model's parameters; typically, independence among parameters is also assumed (e.g., Bertsimas 1992, Zhou and Liu 2003). We will show how B&P can solve SFLP in this situation, too. Furthermore, we will solve it exactly,

that is, with the same certitude that prevails in deterministic mixed-integer programs. This contrasts with alternative solution procedures that provide only probabilistic or asymptotic guarantees for solution quality (e.g., Carøe and Tind 1998, Sen and Higle 2000, Ahmed and Sahinidis 2003).

Solution methods for two-stage stochastic programs (TSSPs) with mixed-integer variables, having a scenario representation of uncertainty, are typically based on branch-and-cut techniques applied to the deterministic mixed-integer problems, such as the integer L-shaped decomposition from Laporte and Louveaux (1993), and the decomposition methods from Carøe and Tind (1998) and from Sen and Higle (2000). These decompositions will suffer if the original formulation of the model has a poor continuous relaxation. In contrast, B&P solves a column-oriented reformulation of a model, also by a form of decomposition, but that formulation will normally have a much tighter relaxation than the original model.

Chapter II has demonstrated substantial improvements in the performance of B&P algorithms on deterministic problems. This success leads us to explore B&P for solving mixed-integer TSSPs. To our knowledge, only Lulli and Sen (2004) and Damodaran and Wilhelm (2004) have previously applied B&P to the stochastic-programming arena, specifically, for a multi-stage stochastic problem. (However, see Shiina and Birge 2004 and Singh et al. 2004, who investigate column generation to solve SMIPs, without using a formal B&P algorithm). Their approach is different from ours, however, because they solve a pricing subproblem for each scenario and leave non-anticipativity constraints in their master problem. In contrast, our master problem contains only first-stage variables—the issue of non-anticipativity constraints in the master problem is consequently moot—and our column-generation subproblems form stochastic programs in and of themselves. Of course, these subproblems solve more quickly than the original, full-scale model.

The next section presents a model for a class of mixed-integer TSSPs and shows how to convert that model to a column-oriented formulation. Several models from the literature provide concrete examples for which the conversion applies. Section C presents the SFLP using a scenario representation for the uncertainty, describes how to

solve it with B&P, and provides computational results. Section D presents a version of SFLP where random parameters take on continuous distributions, describes how to solve it with B&P, and provides computational results. Finally, section E summarizes the work and suggests areas for future research.

B. GENERAL METHODOLOGY

We assume the existence of a TSSP of the following form (see Birge and Louveaux 1997):

Formulation (TSSP)

$$\min_{\mathbf{x}} \sum_{i \in I} \left(\mathbf{c}_i \mathbf{x}_i + E_{\tilde{\boldsymbol{?}}_i} [h_i(\mathbf{x}_i, \tilde{\boldsymbol{?}}_i)] \right) \tag{16}$$

$$\text{s.t.} \sum_{i \in I} A_i \mathbf{x}_i = \mathbf{b} \tag{17}$$

$$\mathbf{x}_i \in X_i \ \forall i \in I \tag{18}$$

$$\text{where} \quad h_i(\mathbf{x}_i, \tilde{\boldsymbol{?}}_i) = \min_{\mathbf{y}_i} \tilde{\mathbf{p}}_i \mathbf{y}_i \tag{19}$$

$$\text{s.t} \quad \tilde{D}_i \mathbf{y}_i \geq \tilde{\mathbf{d}}_i - \tilde{T}_i \mathbf{x}_i \tag{20}$$

$$\mathbf{y}_i \in Y_i, \tag{21}$$

where $\tilde{\boldsymbol{?}}_i$ is a random vector, $\tilde{\boldsymbol{?}}_i \equiv \text{vec}(\tilde{D}_i, \tilde{T}_i, \tilde{\mathbf{d}}_i, \tilde{\mathbf{p}}_i)$. The sets X_i and Y_i contain, at a minimum, non-negativity constraints, but they may also include integrality requirements. $E_{\tilde{\boldsymbol{?}}_i}$ denotes the mathematical expectation with respect to the random vector $\tilde{\boldsymbol{?}}_i$. $\sum_{i \in I} E_{\tilde{\boldsymbol{?}}_i} [h_i(\mathbf{x}_i, \tilde{\boldsymbol{?}}_i)]$ is called the *recourse function* and matrices $\tilde{D}_i$ are often referred to as *recourse matrices* (Walkup and Wets 1967). We further assume that the model possesses the property of *relatively complete recourse* (Rockafellar and Wets 1976), which implies that for any $\mathbf{x}_i \in X_i$, an optimal solution $\mathbf{y}_i$ to constraints (20) and (21) can always be found. We note that $\tilde{D}_i = I$ in some of our examples, implying the property of *simple*

recourse (Beale 1955, Wets 1966). However, this is not an inherent requirement of this class of problems.

The key feature of this model is that the recourse function decomposes by "subproblem" *i*. The general notation disguises some important applications where, for instance, each of the $\tilde{\mathbf{d}}_i$ represent a single, global demand vector, but this should be clarified through the examples.

Now, we further assume that elements of vectors $\mathbf{x}_i$ are integer and the sets X_i are bounded. The principles of Dantzig-Wolfe decomposition then apply (Dantzig and Wolfe 1960), as extended to integer programming by Appelgren (1969). (See Wolsey 1998, section 11.2, for a complete discussion.) To this end, let $\hat{\mathbf{x}}_i^k \in X_i$, $k \in K_i$, denote the enumerated first-stage solutions for subproblem *i*; because of relatively complete recourse, $E_{\tilde{\mathbf{?}}_i}\left[h_i(\hat{\mathbf{x}}_i^k, \tilde{\mathbf{?}}_i)\right]$ is well defined for all such $\hat{\mathbf{x}}_i^k$. The index sets K_i are finite, but normally extremely large. Now, because of the special structure, we can embed $E_{\tilde{\mathbf{?}}_i}[h_i(\hat{\mathbf{x}}_i^k, \tilde{\mathbf{?}}_i)]$, the decomposed recourse function, into the column costs of a column-oriented formulation for TSSP:

Column-Oriented Two-Stage Stochastic Program (CTSSP)

Indices

$i \in I$ subproblems

$k \in K_i$ feasible solutions from X_i

Data

$\hat{\mathbf{x}}_i^k$ the k^{th} feasible solution; $\hat{\mathbf{x}}_i^k \in X_i$

$\mathbf{c}_i$ the first-stage cost for the i^{th} subproblem

Decision variables

l_i^k 1 if $\hat{\mathbf{x}}_i^k \in X_i$ is selected, and 0 otherwise

Formulation (CTSSP)

$$\min_{?} \sum_{i\in I}\sum_{k\in K_i} \left(\mathbf{c}_i\hat{\mathbf{x}}_i^k + E_{\tilde{?}_i}[h(\hat{\mathbf{x}}_i^k, \tilde{\mathbf{?}}_i)]\right) l_i^k \tag{22}$$

$$\text{s.t.} \quad \sum_{i\in I}\sum_{k\in K_i} \left(A_i\hat{\mathbf{x}}_i^k\right) l_i^k = \mathbf{b} \tag{23}$$

$$\sum_{k\in K_i} l_i^k = 1 \qquad \forall\, i\in I \tag{24}$$

$$l_i^k \in \{0,1\} \quad \forall\, i,k\,. \tag{25}$$

CTSSP above can be applied when first-stage variables are general integers, but binary variables would be typical, and we assume that restriction from hereon for simplicity. Second-stage variables may be continuous, integer or both.

We next provide examples of how this reformulation technique applies to some problems from the literature. We supply only short descriptions of the problems, and we ask to the reader to refer to the references for more details.

1. Elastic Generalized Assignment Problem
(Brown and Graves 1981, Appleget and Wood 2000)

In the *elastic generalized assignment problem* (EGAP), the objective is to minimize the cost of assigning capacity-consuming tasks $j \in J$ to capacitated agents $i \in I$, so that (*i*) each task is assigned to exactly one agent, and (*ii*) the total capacity assigned to agent *i* does not exceed his capacity u_i unless an appropriate per-unit penalty p_i is paid. If the capacity required by agent *i* to complete task *j*, is a random variable $\tilde{w}_{ij}$, and the direct cost of that assignment is c_{ij}, then the stochastic model we wish to solve is (Spoerl and Wood 2003):

SEGAP

$$\min_{\mathbf{x}} \sum_{i\in I}\left(\sum_{j\in J} c_{ij}x_{ij} + E_{\tilde{\mathbf{w}}_i}\left[h_i(\mathbf{x}_i, \tilde{\mathbf{w}}_i)\right]\right) \tag{26}$$

$$\text{s.t.} \quad \sum_{i\in I} x_{ij} = 1 \qquad \forall j\in J \tag{27}$$

$$x_{ij} \in \{0,1\} \qquad \forall i\in I, \forall j\in J \tag{28}$$

$$\text{where} \quad h_i(\mathbf{x}_i, \tilde{\mathbf{w}}_i) = \min_{\mathbf{y}} p_i y_i \tag{29}$$

$$\text{s.t} \quad y_i \geq \sum_{j \in J} \tilde{w}_{ij} x_{ij} - u_i \tag{30}$$

$$y_i \geq 0 . \tag{31}$$

Here, x_{ij} equals 1 if task *j* is assigned to agent *i* and is 0 otherwise, $\mathbf{x}_i$ is the vector of assignments for agent *i*, and y_i represents the amount by which agent *i*'s capacity is exceeded. $E_{\tilde{\mathbf{w}}_i}[h_i(\mathbf{x}_i, \tilde{\mathbf{w}}_i)]$ thus represents the expected capacity-violation penalty for agent *i*. Note also that the "capacity-consumption vector" $\tilde{\mathbf{w}}_i$ represents $\tilde{T}_i$ in TSSP, and if capacity consumption depends on the task, not the agent, then $\tilde{T}_i$ might be replaced by a single matrix $\tilde{T}$.

Now, the conversion of SEGAP to a column-oriented formulation is straightforward:

Column-Oriented Formulation for SEGAP (CSEGAP)

Indices

$i \in I$ agents

$j \in J$ tasks

$k \in K_i$ assignments of tasks to a given agent *i*

Data

$\hat{x}_{ij}^k$ 1 if task *j* is assigned to agent *i* in the k^{th} assignment of tasks to agent *i*

$\hat{c}_i^k$ expected cost of the k^{th} assignment of tasks to agent *i* ($\hat{c}_i^k = \mathbf{c}_i \hat{\mathbf{x}}_i^k + E_{\tilde{\mathbf{w}}_i}[h_i(\hat{\mathbf{x}}_i^k, \tilde{\mathbf{w}}_i)]$)

Decision variables

l_i^k 1 if the k^{th} assignment of tasks to facility *i* is chosen, and 0 otherwise

Formulation (CSEGAP)

$$\min_{?} \sum_{i \in I} \sum_{k \in K_i} \hat{c}_i^k l_i^k \tag{32}$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{k \in K_i} \hat{x}_{ij}^k l_i^k = 1 \qquad \forall j \in J \tag{33}$$

$$\sum_{k \in K_i} l_i^k = 1 \qquad \forall i \in I \tag{34}$$

$$l_i^k \in \{0,1\} \qquad \forall i \in I, k \in K_i . \tag{35}$$

Constraints (33) guarantee that each task *j* will be assigned to exactly one agent, and constraints (34) are the convexity constraints for each agent *i*.

2. Time-Constrained Routing and Scheduling
(Desrosiers et al. 1995, Ribeiro and Soumis 1994)

One subset of routing and scheduling problems is the *vehicle routing problem with time windows* (VRPTW). Such a problem describes a fleet of vehicles required to visit a set of customers, each customer being represented by a node in a network. The vehicles have restricted capacities and limited windows of time during which they should perform customer visits. The goal is to assign each customer to a vehicle, such that the sum of capacities required for each customer does not exceed the vehicle's capacity and such that each customer can be visited during the customer-specified time window. The column-oriented formulation for this problem looks exactly like CSEGAP, equations (32)-(35), with indices, parameters and variables appropriately redefined. The parameters $\hat{\mathbf{x}}_i^k$, $k \in K_i$, now represent potential routes (sets of deliveries to customers) for vehicle *i*, each of which covers a subset of the customer set *J*. The recourse function includes expected penalties for violating time windows for deliveries, expected penalties for exceeding a maximum travel time, etc. (Kenyon and Morton 2003).

3. Crew Scheduling (Vance et al. 1997, Day and Ryan 1997)

Following the description in Vance et al. (1997), an airline crew-scheduler wishes to minimize the cost assigning of flight crews to flights in a fixed schedule of flights. *Crew pairings* define feasible trip itineraries that can be assigned to some crew. Each crew pairing consists of a sequence of flights that starts and ends at a home base, respects limits on work hours, allows times for breaks, and satisfies numerous other restrictions.

Set-partitioning models are the norm for this type of problem (e.g., Vance et al. 1997, Day and Ryan 1997), and these fit a simplified form of CSSTP in which I is a singleton, **b** becomes a vector of 1s corresponding to flights that must be covered by crews, and convexity constraints (24) are eliminated; the columns $A_i\hat{\mathbf{x}}_i^k$ represent sets of flights a single crew can feasibly cover, i.e., pairings.

However, "home-base constraints" may need to be enforced (Butchers et al. 2001, Medard and Sawhney 2004), and these simply modify (24) to

$$\sum_{k \in K_i} l_i^k \leq u_i \qquad \forall i \in I\,, \tag{36}$$

where I denotes the set of home bases, u_i denotes the number of crews available at $i \in I$, and the index set K_i now represents potential pairings for crews based at i. For the recourse function, we suggest a probabilistic variant on the function that Ehrgott and Ryan (2003) use to penalize schedules that do not allow adequate time for crews to switch aircraft. Their function is based on averaged historical information, but could be modified to represent a penalty function integrated over empirical or fitted delay distributions.

4. Origin-Destination Integer Multi-Commodity Flow Problem
(Barnhart et al. 2000)

This model is a restricted version of the linear multicommodity flow problem (Ahuja et al. 1993, chapter 17), where each commodity must be shipped from its origin to its destination using only a single path. Therefore, its formulation resembles the well-known path-oriented, i.e., column-oriented formulation, for the linear multicommodity flow problem (e.g., Ford and Fulkerson 1958, Ahuja et al. 1993, section 17.5, Bazaraa et al. 1990, section 12.4), but with binary variables. The binary variables l_i^k for this model represent the kth path for commodity i, and $l_i^k = 1$ indicates the decision to ship all units of commodity i by path k. The paths are represented by arcs and the sum of all commodities flowing on all commodities over each arc must not exceed that arc's

capacity. Constraints (23) will handle these constraints by (*i*) adding slack variables, (*ii*) letting b_j represent the capacity of arc *j*, and (*iii*) by defining $a_{ij}\hat{x}_i^k$ to be the amount of arc *j*'s capacity consumed by path *k* of commodity *i*. The convexity constraints guarantee that only a single path, with appropriate origin and destination, is selected for each commodity. For a communications network, say, each component of the recourse function $E\left[h_i(\mathbf{x}_i, \tilde{\mathbf{?}}_i)\right]$ might represent an expected, path-dependent penalty based on uncertain link availability (Girard and Sansó 1998) or uncertain "hop delay" (e.g., Papagiannaki et al. 2003) which is independent of congestion.

C. SOLVING A STOCHASTIC FACILITY LOCATION PROBLEM BY BRANCH AND PRICE

1. A Stochastic Facility Location Problem with Sole Sourcing

A standard, deterministic, facility-location problem aims to identify the best locations for capacitated production facilities, which will ship to established customers to meet the customers' demands for some product. The mathematical model must find the best trade-off between variable and fixed costs (Laporte et al. 1994): More open facilities leads to lower shipping (variable) costs because plants are closer to customers, on average; on the other hand, opening more facilities means more facility-installation (fixed) costs will be incurred. The deterministic model typically assumes that all costumer demands will be completely satisfied, and, in the version we build upon, a unique facility satisfies each customer's demand. This latter assumption is known as *sole-sourcing*, and the resulting model is called the (deterministic) capacitated facility-location problem with sole-sourcing (FLP) (Barcelo and Casanova 1984).

Assume now that some uncertainty in the data arises in the nominally deterministic FLP: Does a manufacturer really know what his demands, capacities and costs will be in the future? Let us represent that uncertainty through a discrete set of scenarios indexed by *s*, with $\mathbf{c}^s, \mathbf{d}^s$ and $\mathbf{u}^s$ representing shipping costs, customer demands and facility capacities in each scenario, respectively. For simplicity, we assume that if the aggregate demand for a facility exceeds the facility's capacity to produce, the facility

pays a penalty based on the unsupplied amount. This model is reasonable if the "unsatisfied demand" is actually satisfied by the relevant facility acquiring extra product from an outside supplier and shipping it to customers as needed. We are now ready to present a formulation for the SFLP:

Stochastic Facility Location Problem with Sole-Sourcing (SFLP)

Indices

$i \in I$ potential facility locations

$j \hat{I} J$ customers

$s \hat{I} S$ scenarios

Data [units]

$\bar{c}_{ij}$ average total cost for supplying the whole demand of customer j from facility i [dollars]

f_i fixed cost for installing a facility at location i [dollars]

d_j^s demand of customer j, under scenario s [tons]

u_i^s capacity of the facility i, under scenario s [tons]

p_i^s penalty for each unit of unmet demand for facility i under scenario s [dollars/tons]

f_s probability of scenario s

Decision variables [units]

x_{ij} 1 if customer j is assigned to facility i, and 0 otherwise

x_i' 1 if facility i is opened, and 0 otherwise

y_i^s amount of unmet demand for facility i, under scenario s [tons]

Formulation (SFLP)

$$\min_{\mathbf{x},\mathbf{x}',\mathbf{y}} \sum_{i\in I} f_i x_i' + \sum_{i\in I}\sum_{j\in J} \bar{c}_{ij} x_{ij} + \sum_{s\in S}\sum_{i\in I} f_s p_i^s y_i^s \qquad (37)$$

s.t.

$$\sum_{i\in I} x_{ij} = 1 \qquad \forall j \in J \qquad (38)$$

$$-x_i' + x_{ij} \le 0 \qquad \forall i \in I,\ j \in J \qquad (39)$$

$$\sum_{j \in J} d_j^s x_{ij} \qquad -y_i^s \leq u_i^s \quad \forall i \in I, s \in S \qquad (40)$$

$$x_{ij} \in \{0,1\} \quad \forall i \in I,\ j \in J \qquad (41)$$

$$x_i' \in \{0,1\} \quad \forall i \in I \qquad (42)$$

$$y_i^s \geq 0 \qquad \forall i \in I, s \in S\,. \qquad (43)$$

This type of formulation is known as the *extensive form* of a stochastic program (Birge and Louveaux 1997, p. 8), because the second-stage variables and constraints are made explicit for all scenarios.

2. A Column-Oriented Formulation for SFLP

Here we describe a column-oriented formulation for SFLP (CSFLP) that fits directly into the format of CTSSP. In this formulation, we use the word *assignment* to represent any collection of customers that are served by the same facility. Actually, the forms of CSFLP and CSEGAP are identical, requiring only redefinition of indices and variables (and definitions from SFLP which will not be repeated):

Column-Oriented Formulation of SFLP (CSFLP)

Indices

$k \ \hat{I}\ K_i$ assignments of customers to a given facility i

Data [units]

$\hat{x}_{ij}^k$ 1 if customer j is assigned to facility i in the k^{th} assignment of customers to facility i

$\hat{c}_i^k$ total expected cost of the k^{th} assignment of customers to facility i ($\hat{c}_i^k = \sum_{j \in J} \overline{c}_{ij} \hat{x}_{ij}^k + \sum_s f_s p_i^s \hat{y}_i^s + f_i$, except the empty assignment has $\hat{c}_i^k = 0$) [dollars]

Decision variables

l_i^k 1 if the k^{th} assignment of customers to facility i is chosen, and 0 otherwise;

Formulation (CSFLP): Same as (32)-(35)

A column-oriented formulation like CSFLP cannot normally be solved directly because it is impossible, or impractical, to create the full set of columns. Therefore, each K_i is replaced by a subset to form a *restricted master problem* (RMP). The solution to the LP relaxation of the RMP (LP-RMP) then yields dual variables, which can be used to identify new columns with favorable reduced cost through one or more *column-generation subproblems*. In the case of CSFLP, for any facility *i*, the following subproblem arises (assuming the RMP contains all empty assignments; see below):

$\mathbf{CSUB}_i(\hat{\mathbf{p}}, \hat{m}_i)$

$$\min_{\mathbf{x}_i, \mathbf{y}_i} \sum_{j \in J} \left(\bar{c}_{ij} - \hat{p}_j \right) x_{ij} + \sum_{s \in S} f_s p_i^s y_i^s + f_i - \hat{m}_i \tag{44}$$

$$\text{s.t.} \quad \sum_{j \in J} d_j^s x_{ij} - y_i^s \le u_i^s \qquad \forall s \in S \tag{45}$$

$$x_{ij} \in \{0,1\} \qquad \forall j \in J \tag{46}$$

$$y_i^s \ge 0 \qquad \forall s \in S, \tag{47}$$

where $\hat{p}_j$ is the optimal dual variable from the LP-RMP associated with constraint (33) for customer *j*, and $\hat{m}_i$ is the optimal dual variable from the LP-RMP for of the convexity constraint (34) associated with facility *i*. By assuming that the customer demands are satisfied entirely with penalties associated with facilities, only the expected values of the random total shipment costs need be considered. Therefore, if $\tilde{c}'_{ij}$ denotes the random shipping cost for unit of customer *j* demand supplied by facility *i*, the total expected shipping cost for facility *i* to customer *j* is $E[\tilde{c}'_{ij}\tilde{d}_j]$, or, if shipping costs and demands are independent, $E[\tilde{c}'_{ij}]E[\tilde{d}_j]$.

The solution of $\text{CSUB}_i(\hat{\mathbf{p}}, \hat{m}_i)$ represents an assignment of customers to a given facility *i*, and its optimal objective function value is the reduced cost of this assignment with respect to the current solution of the LP-RMP. Therefore, a negative reduced cost indicates that this assignment, $\hat{\mathbf{x}}_i^k$, should be translated into a column for the RMP and inserted into it. The assignment's cost in the LP-RMP will be

$\hat{c}_i^k = \sum_{j \in J} \bar{c}_{ij} \hat{x}_{ij}^k + \sum_s f_s p_i^s \hat{y}_i^s + f_i$. The fixed cost f_i appears as a constant because we include all empty assignments $\hat{\mathbf{x}}_i^k = \mathbf{0}$, which have costs $\hat{c}_i^k = 0$, as part of the initial set of columns in the RMP, and all new assignments must incur the fixed cost.

3. Solving the Column-generation Subproblems

The subproblems $\text{CSUB}_i(\hat{\mathbf{p}}, \hat{m}_i)$ are *multi-dimensional knapsack problems* (Weingartner and Ness 1967) with elastic penalties in each dimension; Kleywegt et al. (2002) refer to these as *static stochastic knapsack problems*. We solve them through straightforward branch and bound, except that we add "explicit constraint branching" (Appleget and Wood 2000) by defining the general integer variables g_i and adding the following constraint to each subproblem i:

$$\sum_{j \in J} x_{ij} - g_i = 0 . \tag{48}$$

The variable g_i is an "ECB variable" and receives a higher priority for branching than do the x_{ij}. Intuitively, constraint branching, explicit or implicit, provides a better balanced branch-and-bound enumeration tree, and this tends to reduce total enumeration (see Ryan and Foster 1981).

4. Enhancing Branch and Price

Branch-and-price algorithms (e.g., Savelsbergh 1997, Barnhart et al. 1998, Barnhart et al. 2000) are appearing as complements to the branch-and-cut algorithms, which can be found implemented in practically all commercial MIP solvers. B&P algorithms combine branch-and-bound techniques with column-generation procedures. Achieving good performance with algorithms based on column-generation is difficult (Lübbecke and Desrosiers 2002), but a number of enhancements to the basic procedure can help, e.g., stabilizing the dual variables used for column generation, strong branching. Chapter II has shown that such refinements do, in fact, lead to substantially faster solution times, so this chapter takes advantage of them. We now comment briefly on the enhancements used in this work

By applying duals stabilization, we try to accelerate the column-generation process used to solve the linear relaxation of the CSFLP; we follow Du Merle et al. (1999) for this purpose. The main idea is to use a *dynamic elastic trust region* for the dual variables, such that penalties are applied when the dual values lie beyond the nominal trust region limits. The trust region is dynamic because it is continuously resized and the penalties adjusted based on the most recent values of the dual variables.

Strong branching (Johnson et al. 2000) is another feature used in our B&P algorithm. In order to improve branching efficiency without increasing subproblem complexity, a set of variables that appear attractive for branching purposes, rather than a singleton, is selected, and branching is carried out for each variable selected. Child problems for each branch are created, and fully optimized, and the most attractive branch is followed. "Most attractive" simply implies the branch with minimum objective function value for our implementation. We only consider "strong-branching sets" of cardinality two: This is a small number compared to standard applications of strong branching, but the B&P paradigm requires much more work to re-optimize after branching, so this number must be kept small.

The other enhancements are straightforward and they are explained together with the computational results.

5. Computational Results

We implement our B&P algorithm using software from the COmputational INfrastructure for Operations Research (COIN-OR, or simply COIN), which provides a repository of distinct libraries that can be integrated to build optimization algorithms (Lougee-Heimer 2003). The specific library in COIN that provides the basic framework for a branch-and-price algorithm, called BCP, is originally designed to be executed in a parallel/distributed environment, and the protocol that emulates this environment in a serial one, incurs some computational overhead. The largest portion of this overhead results from the branch-and-bound tree being managed separately from the code that solves the LP-RMPs. We believe that this overhead could be avoided with some additional programming, so we have excluded that overhead from the times we report, denoted *total time* (TT). However, we note that the true CPU time for our

implementations are never more than 10% longer than TT and the mean overhead for all problems solved in this section is 3.1%.

We have implemented our B&P algorithm also using COIN's open solver interface (OSI), coupled with CPLEX 8.0. Thus, the linear relaxation of the RMP and the subproblems are submitted to the CPLEX LP solver and MIP solver, respectively. We carry out all tests on a networked workstation with a 2 GHz Pentium 4 processor and 1 GB of RAM; computing times are measured in CPU seconds. For comparison, we also directly solve the compact formulations of the SFLP problems using CPLEX 8.0. Solution times for this methodology are presented under the label "IP CPX" in the tables below.

We investigate eight groups of problems. Each group is defined by "problem size," meaning "number of facilities - number of customers". The groups used have the following problem sizes: 5-15, 5-30, 8-24, 8-48, 10-30, 10-40, 10-50 and 10-60. And for each problem size, we consider instances with one, ten or fifty scenarios. The one-scenario problem represents a deterministic model. (We investigate the current limits of our technology by expanding the number of scenarios, facilities and customers in section C.7.) Because run times vary somewhat between instances of the same size, we examine five different instances for each combination of problem size and number of scenarios.

To generate the test problems, we first create a reference problem according to the following rules: (*i*) Customer demands d_j are integers from a discrete uniform distribution $U(5,25)$, (*ii*) cost coefficients are integers from $U(15,25)$, (*iii*) facility capacities are $u_i = 0.8\sum_{j\in J} d_j / |I|$, and (*iv*) the fixed cost for each facility is 1.5 times the value of its capacity, u_i, in the reference problem. Chu and Beasley (1997) use rules (*i*) and (*ii*) to generate the "small instances" of the generalized assignment problem in their research. For the stochastic instances, the demands are uniformly distributed within ±20% of their value in the reference problem, while the capacities are ±10%. The additional cost (penalty) a facility *i* pays for a unit of "unmet demand" it satisfies through an outside purchase is $p_i = 0.4\max_{j\in J} c_{ij}$.

Tables 8 and 9 show TT for each of the generated instances, solved by the IP, and by B&P, with and without duals stabilization; we wait until the following section to explore other enhancements to B&P. The values in bold indicate the fastest times among the three algorithms. Parameter settings for the duals stabilization are fixed for all problems tested; we have not attempted to optimize settings for each specific problem. We have arbitrarily set an upper limit of 7,200 seconds on total allowed computation time.

	Problem Sizes (facilities-customers)											
	5-15			**5-30**			**8-24**			**8-48**		
Sc	**IP CPX**	**No Stz**	**Stz**	**IP CPX**	**No Stz**	**Stz**	**IP CPX**	**No Stz**	**Stz**	**IP CPX**	**No Stz**	**Stz**
1	3.5	**0.5**	0.8	2.5	**1.7**	**1.7**	2274.9	**1.1**	14.5	5.8	5.0	**2.7**
1	**0.1**	0.3	0.8	**1.5**	4.1	3.9	*	**2.0**	2.7	3.8	3.4	**2.4**
1	3.2	**0.6**	2.0	**0.9**	1.4	1.9	274.1	**1.6**	1.8	4.5	6.4	**2.7**
1	**0.2**	0.3	0.6	**1.6**	2.1	1.9	299.8	**1.3**	4.4	3.8	3.2	**2.6**
1	3.6	**0.5**	1.4	8.6	**2.4**	2.8	1.8	**0.8**	0.9	5076.1	61.2	**44.6**
10	1.3	**1.1**	1.6	4.2	**3.7**	**3.7**	881.0	**4.8**	9.1	142.4	7.9	**4.2**
10	1.4	**1.0**	1.4	**5.1**	16.6	30.5	2987.5	**4.3**	7.0	39.7	6.7	**5.7**
10	1.0	**0.9**	1.1	**1.2**	2.8	3.0	231.3	**4.8**	5.7	20.6	8.4	**5.2**
10	**0.4**	0.6	1.3	**2.4**	3.2	3.6	16.4	**2.7**	2.9	19.8	6.7	**5.9**
10	1.0	**0.7**	1.4	9.4	**3.6**	6.8	3.9	**1.2**	1.3	*	29.7	**25.8**
50	2.5	**2.3**	3.0	**4.9**	10.1	10.2	1637.2	45.5	**29.6**	455.6	**40.0**	58.5
50	3.1	**2.3**	4.4	**11.0**	11.1	11.1	5579.0	8.7	**7.6**	45.7	20.2	**14.0**
50	2.8	**2.3**	2.7	**3.4**	7.8	8.5	1081.6	8.0	**7.6**	53.4	25.0	**15.9**
50	**1.3**	1.4	3.1	**4.3**	8.6	10.1	57.4	**5.8**	6.0	129.1	24.3	**20.9**
50	3.9	**2.7**	3.3	12.2	12.1	**10.7**	6.6	**4.2**	**4.2**	990.2	**80.1**	114.6

Table legend:
Sc: Number of scenarios
IP CPX: CPLEX MIP solver with presolver on
No Stz: Branch and price without duals stabilization
Stz: Branch and price using duals stabilization
* Problem not solved to optimality after 7,200 CPU seconds.

Table 8. Total time (TT) in CPU seconds for randomly generated SFLPs. The solution methods used are (*i*) branch and bound on the extensive formulation using the CPLEX solver, (*ii*) B&P without duals stabilization, and (*iii*) B&P with duals stabilization. Numbers in bold indicate the fastest of the three competing solution methods.

	Problem Sizes (facilities-customers)											
	10-30			10-40			10-50			10-60		
Sc	**IP CPX**	**No Stz**	**Stz**	**IP CPX**	**No Stz**	**Stz**	**IP CPX**	**No Stz**	**Stz**	**IP CPX**	**No Stz**	**Stz**
1	*	**16.7**	17.0	15.0	3.0	**2.2**	*	**67.5**	143.0	21.5	6.8	**3.8**
1	3.5	**1.1**	**1.1**	7.0	**2.1**	**2.1**	*	53.8	**50.8**	22.3	11.0	**5.3**
1	44.8	**1.0**	1.5	13.6	2.8	**2.1**	1656.8	9.7	**7.2**	14.8	8.3	**4.4**
1	1.4	**1.3**	1.5	*	36.2	**10.2**	*	116.8	**99.1**	13.5	11.9	**5.2**
1	*	**5.3**	19.5	6.4	3.2	**1.7**	50.0	5.5	**4.1**	23.8	13.4	**4.3**
10	*	**10.3**	12.9	2322.7	6.0	**4.0**	*	20.6	**13.9**	37.2	16.7	**8.0**
10	5.2	2.4	**1.8**	860.1	5.2	**3.8**	*	**310.4**	424.2	3163.1	30.2	**22.4**
10	7.0	3.3	**2.6**	262.1	5.6	**4.1**	*	18.0	**14.1**	140.5	23.1	**14.5**
10	5.0	**2.0**	2.1	*	**53.6**	77.2	*	19.7	**11.7**	45.6	14.2	**10.8**
10	*	**15.0**	29.4	612.7	**5.1**	11.2	*	**20.5**	41.2	30.7	19.7	**9.6**
50	*	16.2	**15.6**	3347.0	20.6	**14.2**	*	**99.1**	127.2	154.6	37.7	**18.8**
50	12.6	**6.5**	7.1	1893.9	14.5	**11.6**	*	37.9	**22.9**	*	60.0	**50.7**
50	51.7	**7.7**	10.7	573.0	14.8	**11.3**	*	**112.3**	171.7	132.3	47.9	**39.9**
50	16.6	6.7	**6.0**	*	30.8	**27.3**	*	33.9	**27.9**	97.1	34.4	**21.5**
50	*	**26.0**	115.0	3559.3	17.3	**12.3**	*	**72.7**	111.5	213.3	39.7	**22.2**

Table legend: same as Table 8

Table 9. Total time (TT) in CPU seconds for randomly generated SFLPs. The solution methods used are (*i*) branch and bound on the extensive formulation using the CPLEX solver, (*ii*) B&P without duals stabilization, and (*iii*) B&P with duals stabilization. Numbers in bold indicate the fastest of the three competing solution methods.

6. Discussion

One might be concerned that B&P requires so much overhead that it could not be effective for small problems. However, the problems in Table 8 have only five or eight potential facilities, and IP outperforms B&P only in problems with five facilities, and then only by a small amount. Moreover, average solution times for B&P are at least an order of magnitude faster than IP, and IP cannot even solve two of the problems within the time limit of 7,200 CPU seconds. Even for small problems, B&P is the right choice.

Table 9, which covers problems with 10 facilities, clearly shows that B&P solution times are more stable and suffer less when the number of scenarios is increased. We can see that 23 problems out of 60 could not be solved by IP within 7,200 CPU seconds, and IP never outperforms B&P. Moreover, B&P can be orders of magnitude faster than solving the original problem for single-scenario problems, i.e., for deterministic problems.

7. Pushing the Limits

We now test our approach for solving problems with a larger numbers of facilities, customers and scenarios. The problem sizes (facilities-customers) are 10-60, 20-60, 15-80 and 20-100 here. Camm et al. (1997) solve a facility-location model for a commercial application with 17 potential facilities and 123 customer zones, so our largest problem is roughly the same size as at least one real-world problem. A larger number of scenarios will be useful when the solution method is based on sample average approximations (Mak et al. 1999).

Number of scenarios	**Problem Size (facilities-customers)**			
	10-60	20-60	15-80	20-100
50	36.4	64.6	47.2	137.2
50	35.2	55.7	65.1	257.2
50	26.9	58.1	54.6	108.2
50	35.9	62.1	44.0	106.0
50	26.7	53.4	106.6	154.9
100	48.2	136.8	132.7	229.6
100	68.4	96.5	108.0	829.6
100	56.2	109.4	238.3	165.7
100	73.0	123.9	75.6	150.0
100	51.0	90.4	76.3	257.0
200	134.6	245.3	181.5	513.0
200	131.3	199.4	302.2	1294.6
200	103.7	168.9	762.1	260.7
200	134.0	184.5	157.1	272.2
200	103.8	174.4	156.5	555.0
300	154.8	374.4	305.4	507.8
300	248.7	305.8	391.2	817.0
300	144.6	250.0	654.1	535.4
300	164.0	320.9	493.2	493.2
300	210.8	276.0	274.4	687.1

Table 10. Total time (TT) in CPU seconds for randomly generated SFLPs with scenario uncertainty. This table explores the computational limits of our current B&P implementation with duals stabilization.

We can see here that solution times tend to increase only slowly, perhaps linearly, with the number of scenarios. Thus, the number of scenarios does not seem to be a

strongly limiting factor with the B&P methodology. This bodes well for solving an SMIP through sampled approximating problems, since the probability of identifying the optimal solution for a discrete TSSP increases exponentially with the number of sampled scenarios (Kleywegt et al. 2002).

D. SOLVING A SPECIAL CASE OF SFLP EXACTLY

Here we wish to investigate a special case of the SFLP in which a number of the parameters are independent, continuously distributed random variables. Models making similar assumptions appear frequently in the stochastic programming literature (e.g., Louveaux and Peeters 1992, Laporte et al. 1994); however, such models are rarely solved exactly as we shall solve SFLP. Even if the reader believes such assumptions are unreasonable in a real-world facility-location problem, it is instructive to see that exact solutions can be achieved for such a model in this column-oriented framework. Perhaps these assumptions will be more appropriate in another application of our methodology.

Consider now a random vector $\tilde{\mathbf{?}} = \text{vec}(\tilde{\mathbf{c}}, \tilde{\mathbf{d}}, \tilde{\mathbf{p}})$ whose elements represent shipping costs, customer demands and unmet-demand penalties, respectively. The column-oriented formulation for SFLP with these random parameters resembles equations (32)-(35), with the following modified definitions:

Data [units]

$\tilde{c}_{ij}$ shipping cost for supplying all of customer j's demand from facility i [dollars]

$\bar{c}_{ij}$ expected shipping cost $E[\tilde{c}_{ij}]$ [dollars]

$\tilde{d}_j$ demand from customer j [tons]

u_i capacity of facility i [tons]

$\tilde{p}_i$ unit penalty for unmet demand that must be covered by facility i [dollars/tons]

$\bar{p}_i$ expected unit penalty $E[\tilde{p}_i]$ [dollars/tons]

$\hat{c}_i^k$ total expected cost of the k^{th} assignment of customers to facility i, $\hat{c}_i^k = \bar{\mathbf{c}}_i \hat{\mathbf{x}}_i^k + E_{\tilde{?}_i}[h_i(\hat{\mathbf{x}}_i^k, \tilde{\mathbf{?}}_i)])$ [dollars] or, more precisely,

$$\hat{c}_i^k = \begin{cases} 0 & \text{if } \hat{\mathbf{x}}_i^k = 0 \\ \overline{\mathbf{c}}_i \hat{\mathbf{x}}_i^k + \overline{p}_i E\left[\left(\sum_{j \in J} \tilde{d}_j \hat{x}_{ij}^k - u_i\right)^+\right] + f_i & \text{otherwise.} \end{cases} \tag{49}$$

1. Normally Distributed Demands

For the special-case model, each customer *j* has demand $\tilde{d}_j$ which is independent and normally distributed with mean m_j and variance v_j, i.e., $\tilde{d}_j \sim N(m_j, v_j)$. Penalties can be continuously distributed random variables that are independent of demands. In fact, with the assumption of independence, they appear in the objective function only through their means, so they may have arbitrary distributions as long as they have finite means. For simplicity, we represent penalties through their vector of means $\overline{\mathbf{p}}$.

For simplicity in exposition, we also assume that the means m_j and the variances v_j are integers, but this is only for simplicity in exposition. The reader will see that our techniques easily extend to means $a_j m_j$ and variances $b_j v_j$, where a_j and b_j are positive scale parameters, and m_j and v_j represent integers running from 0 to some finite upper bound. Given the efficiency of the dynamic-programming solution procedure that uses m_j and v_j, the scale parameters can be quite small, and thus a wide range of actual mean and variance combinations can be closely approximated.

The RMP for this model does not change from the column-oriented formulation CSFLP presented in section C.2. To solve this special case exactly, we will solve the subproblems corresponding to the formulation (44)-(47) exactly. Given equation (49), the subproblem associated with facility *i* is this static stochastic knapsack problem (Kleywegt et al. 2002):

$$z^* = \min_{x_{ij} \in \{0,1\}\ \forall j \in J} \left\{ \sum_{j \in J} \left(\overline{c}_{ij} - p_j\right) x_{ij} + \overline{p}_i E\left[\left(\sum_{j \in J} \tilde{d}_j x_{ij} - u_i\right)^+\right] \right\} + f_i - m_i, \tag{50}$$

where $s^+ \equiv \max\{0, s\}$. Evaluating $E\left[\left(\sum_{j\in J} \tilde{d}_j x_{ij} - u_i\right)^+\right]$ is easy, because we know that (see Kleywegt et al. 2002)

$$E\left[\tilde{w}(m,v)^+\right] = m\Phi\left(\frac{m}{\sqrt{v}}\right) + \sqrt{\frac{v}{2\mathrm{p}}}\exp\left(-\frac{m^2}{2v}\right), \tag{51}$$

where $\tilde{w}(m,v) \sim N(m,v)$.

Ignoring the constraint-violation penalties for the moment, we apply dynamic programming to evaluate the function $g(|J|,m,v)$ defined as

$$g(|J|,m,v) = \min \sum_{j\in J} (\bar{c}_{ij} - p_j)x_{ij} \tag{52}$$

$$\text{s.t.} \quad \sum_{j\in J} m_j x_{ij} \le m \tag{53}$$

$$\sum_{j\in J} v_j x_{ij} = v \tag{54}$$

$$x_{ij} \in \{0,1\}, \tag{55}$$

for $m = 0,...,m_{\max}$, and $v = 0,...,v_{\max}$, where $m_{\max} = \sum_{j\in J} m_j$ and $v_{\max} = \sum_{j\in J} v_j$. The forward recursion is:

Initialization:

$g(j,m,v) = 0$ for $j = 0, m = 0, v = 0$;

$g(j,m,v) = +\infty$ for $j = 0, m \neq 0, v \neq 0$;

Recursion:

$$g(j,m,v) = \min_{\substack{j=1,...,|J| \\ m=1,...,m_{\max} \\ v=1,...,v_{\max}}} \left\{g(j-1,m,v), \bar{c}_{ij} - p_j + g(j-1,m-m_j,v-v_j)\right\}. \tag{56}$$

This recursion is similar to that for a two-dimensional knapsack problem, but for a given *m*, the objective value g_i does not depend on *v*. This variance will be used in a final calculation, however.

Now, since an assignment of customers to a facility *i*, $\hat{\mathbf{x}}_i$, yields an aggregate demand with distribution $N\left(\sum_{j\in J} m_j \hat{x}_{ij}, \sum_{j\in J} v_j \hat{x}_{ij} \right)$, the optimal objective value for (50) will be

$$z^* = \min_{\substack{m=1,\ldots,m_{\max} \\ v_{\max}=1,\ldots,v_{\max}}} \left\{ g\left(|J|, m, v\right) + \bar{p}_i E\left[\tilde{w}(m-u_i, v)^+ \right] \right\} + f_i - m_i . \qquad (57)$$

For the case where the facilities capacities $\tilde{u}_i$ are also independent and normally distributed, and independent of the customer demands, the method just described also qualifies with a slight modification: The expectation in equation (57) changes to $E\left[\tilde{w}\left(m - E[\tilde{u}_i], v + Var(\tilde{u}_i)\right)^+ \right]$.

2. Extensions

The methodology described above will fit other problems, but numerical integration may be required. In a general case, suppose each demand can be defined by $\tilde{d}_j = \sum_{k=1}^{K_j} \tilde{r}_{jk} + m_j$ where all $\tilde{r}_{jk}$ have a common distribution with mean 0 and variance *v*, and all m_j are positive integers; all u_i are deterministic, here. Then, any aggregate demand $\sum_{j\in J} \tilde{d}_j x_{ij}$ can be described through its integer mean $\sum_{j\in J} m_j x_{ij}$, and its scaled integer variance $v \sum_{j\in J} K_j$. Thus, the recursion (56) applies with appropriate adjustments for the scaled variances. And then, $E\left[\left(\sum_{j\in J} \tilde{d}_j x_{ij} - u_i \right)^+ \right]$ can be easily computed by numerical

integration over the distribution of $\sum_{j \in J} \tilde{d}_j x_{ij}$, which is completely defined by its bounded integer mean, its bounded scaled-integer variance, and the common distribution for $\tilde{r}_{jk}$.

3. Computational Results for Normally Distributed Demands

To build the test instances for this special case, we select all problems with one scenario solved in section C, and assume that their demands represent expected values of the normally distributed demands used here. Given the expected values for demands, we generate the variances as discrete uniform random variables on [1, V_j], where V_j is the maximum value that assures $P(\tilde{d}_j < 0) \leq 0.001$ (e.g., Spoerl and Wood 2003).

Table 10 presents computational results. The columns represent computational times using different combinations of enhancements applied to the B&P algorithm; the enhancements are cumulative, from left to right. The enhancement "Solve one subproblem" means that as soon as a subproblem encounters a favorable column, no more subproblems are solved on the current column-generation iteration. The column is added to RMP and its LP relaxation is re-solved. The order of the subproblems is then randomized to ensure that algorithm does not focus on one subproblem at the expense of others. "Delete poor columns" means that at every specified number of column-generation iterations, all columns with reduced cost above a given threshold are deleted from the RMP. The parameter settings remain constant for all problems.

Problem Size		Basic B&P	+Duals stabilization	+Strong branching	+Solve one subproblem	+Delete poor columns
Facilities	Customers					
5	15	4.3	7.0	4.7	**2.6**	2.8
		2.6	2.5	2.3	**2.0**	**2.0**
		2.0	1.2	1.2	**1.0**	**1.0**
		4.1	3.8	3.8	**3.2**	3.3
		3.6	3.4	3.4	**2.0**	**2.0**
5	30	79.1	62.9	61.9	**42.2**	42.4
		46.4	34.1	34.5	**21.3**	21.5
		55.7	68.4	68.9	**23.9**	24.0
		60.3	48.6	48.2	33.0	**32.8**
		116.3	153.1	131.1	75.0	**74.7**
8	24	18.4	13.6	13.3	**9.9**	**9.9**
		17.2	12.7	12.3	**8.3**	**8.3**
		42.2	25.0	**24.7**	33.3	32.9
		14.9	8.7	8.6	**7.2**	**7.2**
		35.8	24.0	24.0	**16.2**	**16.2**
8	48	143.2	77.9	78.2	**58.8**	**58.8**
		158.7	90.1	89.0	71.6	**70.4**
		167.8	116.8	108.3	87.6	**83.4**
		140.4	84.8	85.0	70.3	**68.7**
		165.7	117.0	133.7	85.8	**80.8**
10	30	152	218	211	79	**78**
		80	57	57	**34**	**34**
		111	59	60	**39**	40
		118	64	64	**38**	**38**
		59	78	78	**30**	**30**
10	40	151	88	88	**58**	59
		277	157	156	**112**	113
		387	192	192	179	**177**
		309	184	171	248	**152**
		261	168	168	**118**	119
10	50	667	342	344	**291**	317
		768	**412**	655	449	447
		932	1100	1165	753	**711**
		658	650	652	**385**	417
		644	288	287	233	**221**
10	60	1896	1166	**1160**	1742	1900
		1617	**933**	941	1339	2648
		2404	**1615**	1618	1885	2557
		1824	1358	**1339**	3086	2612
		2157	1109	1101	**838**	859

Table 11. Total time (CPU seconds) to solve SFLP with B&P when demands are independent and normally distributed. The columns represent the enhancements applied in the B&P algorithm, cumulatively from left to right. The times in bold font indicate the fastest solution time for each problem among the different combinations of enhancements, i.e., across each row.

4. Discussion

The enhancements applied to the special-case instances always improve solution times. Moreover, the solution times tend to be more stable among instances of the same size. The benefits of the enhancements are also more consistent, indicating that "duals stabilization" should be defined as a default option for this special case.

The results displayed in Tables 8 and 11 indicate that B&P performs better on problems with a smaller customers-to-facilities ratio. This also happens in small deterministic integer problems, where that ratio positively is correlated with the number of feasible solutions the problem instances have (Savelsbergh 1997, chapter II of this dissertation). In turn, the number of feasible solutions is positively correlated with the number of columns in the column-oriented model, and the more columns a problem has, the harder it is to solve. We have also tested the enhancements used in this section on problems with scenario uncertainty; however, beyond duals stabilization, our experiments do not show the consistent improvements as seen here. Even among problems of the same size, we have difficulty finding a combination of enhancements that is consistently effective.

E. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

1. Summary

This chapter has proposed column-oriented formulations for a class of two-stage stochastic mixed-integer programs (TSSPs), and has shown how to solve such models using a branch-and-price algorithm (B&P). For demonstration purposes, we use this methodology to solve a stochastic facility-location problem (SFLP) using a scenario representation of uncertainty. In addition, we formulate and solve a special case of SFLP assuming certain continuous distributions for uncertain parameters, and demonstrate how B&P can solve that problem exactly. We also provide examples of other well-known optimization problems whose stochastic versions fit our new solution approach.

Our B&P algorithm is based on the framework provided by the open-source libraries of the COmputational INfrastructure for Operations Research (COIN-OR, or

simply COIN) and CPLEX 8.0 is used as the LP and MIP solvers. Moreover, we demonstrate how the algorithm performance can be improved by duals stabilization and other techniques.

2. Conclusions

This research demonstrates that B&P is an attractive solution alternative for certain mixed-integer TSSPs. For the SFLP with a scenario representation of uncertainty, computational results show that B&P can be orders of magnitude faster than solving the original problem by branch and bound, and this can be true even for single-scenario problems, i.e., for deterministic problems. When solving the SFLP with continuously distributed random parameters, the enhancements to B&P we investigate substantially improve solutions times. However, except for duals stabilization, the effects of the various enhancements are not consistent with the effects observed in the model with scenario uncertainty.

3. Recommendations for Further Work

The B&P approach can be used to solve, at least approximately, TSSPs of the class described in section B, but with more general probability distributions. For instance, sample-average approximations (Mak et. al 1999, Kleywegt et al. 2002), a method of for solving TSSPs, provides probabilistic guarantees on solution quality and is based on repeated solutions of sampled approximating problems. However, a sampled approximating problem is identical to a stochastic program using a scenario representation of uncertainty.

Sampled subproblems can be used to identify favorable columns in a "nearly exact algorithm," too. Suppose that once a subproblem's integer variables are fixed for a column, the expected cost of the column can be estimated extremely accurately through sampling. This certainly holds for the SFLP, where the penalties associated with, say, 10,000 sampled demands for some fixed facility-to-customers assignment can be sampled and averaged in a fraction of a second. For all intents and purposes, that average will exactly equal the expected cost for the column, and an LP-RMP containing such columns

would yield exact dual solutions. The only difficulty with this procedure would be that solution of the sampled subproblem might indicate that a column is favorable, but extended sampling would reveal that it is not. If we solve many sampled subproblems and cannot identify an improving column, then we might become convinced that we have, in fact, solved the LP-RMP. However, a formal procedure will need to be constructed to provide a rigorous "level of conviction."

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SUMMARY, CONTRIBUTIONS AND FUTURE WORK

This research first demonstrates improvements in the performance of branch-and-price algorithms (B&P) for solving integer programs by applying enhancements such as stabilizing dual variables during column-generation, performing strong branching, inserting multiple near-optimal columns from each subproblem, and heuristically improving feasible integer solutions. Subsequently, we propose a new column-oriented formulation for a class of two-stage stochastic mixed-integer programs (TSSPs) and show how the B&P methodology applies to their solution. For demonstration purposes, we use this methodology to solve a stochastic facility-location problem with sole-sourcing (SFLP). The first version uses a scenario representation of uncertain demands, costs and penalties, and the second represents those parameters as independent, continuously distributed random variables. Demands are normally distributed (other possibilities are discussed), but the distributions of other parameters are nearly arbitrary as they can be represented through their means. Both versions of SFLP are solved exactly.

A. CONTRIBUTIONS

This research has advanced the state of the art in integer programming by bringing a number of techniques into the B&P framework, improving its performance without breaking its structure. Stabilizing dual variables during column-generation is clearly the most important technique, reducing considerably the time spent to solve the LP relaxation of the column-oriented B&P master problem (LP-RMP). This technique views the LP-RMP in term of its dual formulation, and applies a dynamic, elastic trust region to its solution (Du Merle 1999). Strong branching (Johnson et al. 2000) is used to successfully in some problems to increase the branching efficiency without making the subproblems more complex. The other features that we fit into the B&P structure are (*i*) inserting multiple near-optimal columns from each subproblem, (*ii*) improving feasible integer solutions heuristically, and (*iii*) culling unprofitable columns from the LP-RMP periodically.

By adding those features, B&P reaches a new level of usefulness as a tool for solving mixed-integer programs.

Our results also show that B&P works well even with small problems that can be readily solved by standard branch and bound. This contrasts with the view that B&P, because of substantial overhead, is suitable only for large problems (Barnhart et al. 1998).

Using the generalized assignment problem for testing in chapter II, this research identifies situations in which B&P tends to perform better than branch and bound on the compact formulation of a problem, and when the opposite tendency arises. In particular, B&P seems to work better on problems with fewer feasible solutions, which tend to be the difficult ones for branch and bound. (Fewer feasible solutions mean that the explicit column-oriented formulation has fewer variables, and problems with fewer variables tend to be easier.) This makes B&P an important complement to the set of potential solution techniques for integer programs.

We have also proposed, although not tested, techniques for modifying the B&P structure to operate with side constraints that would normally be assumed to eliminate the possibility of a B&P solution.

This research also extends the use of B&P to a class of stochastic mixed-integer programs. A stochastic-facility location problem with a scenario representation of uncertainty can be solved orders of magnitude faster with B&P than it can by branch and bound, and this can be true even for single-scenario problems, i.e., for deterministic problems. We also show how to solve certain SFLPs with continuously distributed random parameters, and demonstrate how the enhancements to B&P can reduce solutions times substantially. However, except for duals stabilization, their effects are inconsistent with the effects observed in the scenario-representation model. This general approach is new, and certain classes of problems can be solved exactly, compared to standard methods that typically provide only probabilistic or asymptotic guarantees on solution quality.

We have implemented B&P using the resources from the COmputational INfrastructure for Operations Research (COIN-OR, or simply COIN), which comprises a

set of distinct libraries that can be integrated to build optimization algorithms. The library used is called Branch-Cut-Price (BCP) (Lougee-Heimer 2003), which is a C++ framework for solving mixed-integer programs in parallel on a distributed network of workstations. We have shown that COIN/BCP will run successful under Microsoft Windows, even before the COIN/BCP literature has claimed this ability. We believe that our work will help making COIN's B&P technology accessible to a wider audience.

B. FUTURE WORK

The possibilities for future work abound. With regards to implementation of B&P, the COIN/BCP framework should be simplified to achieve better performance when running in a serial environment (instead of the parallel/distributed environment for which it has been built.) On the other hand, B&P algorithms are highly suitable for running in a parallel/distributed environment (Ralphs et al. 2003), and we look forward to testing our code there. The code conversion should be straightforward since the COIN/BCP library provides the necessary support.

With regards to B&P and stochastic programming, we see three main areas for future work: (*i*) Applying the technique to new models of the problem class we have described here, (*ii*) developing algorithmic variants that provide probabilistic guarantees for the solution quality when subproblems cannot be solved optimally, and (*iii*) extending the application of B&P to a wider class of stochastic models, including multi-stage models.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Ahmed, S., N. V. Sahinidis. 2003. An approximation scheme for stochastic integer programs arising in capacity expansion. *Operations Research* **51** 461–471.

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows.* Prentice Hall, Englewood Cliffs, New Jersey.

Alvarez, R. E. 2004. Interdicting electrical power grids. Masters Thesis, Operations Research Department, Naval Postgraduate School, Monterey, California.

Amini, M. M., M. Racer. 1994. A rigorous computational comparison of alternative solution methods for the generalized assignment problem. *Management Science* **40** 868–890.

Anbil, R., F. Barahona, L. Ladanyi, R. Rushmeier, J. Snowdon. 1999. Airline optimization. *ORMS Today* **26** 26–29.

Anbil, R., R. Tanga, E. L. Johnson. 1992. A global approach to crew-pairing optimization. *IBM Systems Journal* **31** 71–78.

Appelgren, L. H. 1969. A column generation algorithm for a ship scheduling problem. *Transportation Science* **3** 53–68.

Appelgren, L. H. 1971. Integer programming methods for a vessel scheduling problem. *Transportation Science* **5** 64–78.

Appleget, J. A., R. K. Wood. 2000. Explicit-constraint branching for solving mixed-integer programs. M. Laguna, J.L. González-Velarde, eds. *Computing Tools for Modeling, Optimization and Simulation*, Kluwer Academic Publishers, Boston, MA. 243–261.

Barcelo J., J. Casanova. 1984. A heuristic lagrangean algorithm for the capacitated plant location problem. *European Journal of Operational Research*, **15** 212–226.

Barnhart, C., C. A. Hane, P. H. Vance. 2000. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research* **48** 318–326.

Barnhart, C., E. L. Johnson, G. L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations Research* **46** 316–329.

Bazaraa M. S., J. J. Jarvis, H. D. Sherali. 1990. *Linear programming and network flows*. John Wiley & Sons, New York.

Beale, E. M. L. 1955. On minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society*. **17B** 173–184.

Beasley, J. E. 1990. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society* **41** 1069–1072.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* **4** 238–252.

Bertsimas, D. J. 1992. A vehicle routing problem with stochastic demand. *Operations Research* **40** 574–585.

Bertsimas D., J. N. Tsitsiklis. 1997. *Introduction to Linear Optimization*. Athena Scientific Belmont MA

Birge, J. R., F. Louveaux. 1997. *Introduction to Stochastic Programming*. Springer-Verlag, New York.

Brown, G. G., G. Graves. 1981. Real-time dispatch of petroleum tank trucks. *Management Science* **27** 19–32.

Butler, J. C., J. S. Dyer. 1999. Optimizing natural gas flows with linear programming and scenarios. *Decision Sciences* **30** 563–580.

Butchers, E. R., P. R. Day, A. P. Goldie, S. Miller, J. A. Meyer, D. M. Ryan, A. C. Scott, C. A. Wallace. 2001. Optimized Crew Scheduling at Air New Zealand. *Interfaces* **31** 30–56.

Camm J. D., T. E. Chorman, F. A. Dill, J. R. Evans, D. J. Sweeney, G. W. Wegryn. 1997. Blending OR/MS, judgment, and GIS: Restructuring P&G's supply chain. *Interfaces* **27** 128–142.

Carlyle W. M., R. K. Wood. 2003. Near-shortest and k-shortest simple paths. *Networks.* To appear.

Carøe, C. C., J. Tind. 1998. L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming* **83** 451–464.

Cattrysse, D. G., M. Salomon, L. N. Van Wassenhove. 1994. A set partitioning heuristic for the generalized assignment problem. *European Journal of Operational Research* **72** 167–174.

Cattrysse, D. G., L. N. Van Wassenhove. 1992. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research* **60** 260–272.

Chabrier, A. 2003. Heuristic branch-and-price-and-cut to solve a network design problem. Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Montreal, Canada, 8–10 May.

Charnes, A. 1952. Optimality and degeneracy in linear programming. *Econometrica* **20** 160–170.

Chen, Z.-L., S. Li, D. Tirupati. 2002. A scenario-based stochastic programming approach for technology and capacity planning. *Computers and Operations Research* **29** 781–806.

Chu P. C., J. E. Beasley. 1997. A genetic algorithm for the generalized assignment problem. *Computers and Operations Research* **24** 17–23.

COIN in http://www.coin-or.org (accessed July 2004).

CPLEX Optimization 2002, CPLEX Linear Optimizer 7.5 with Mixed Integer and Barrier Solvers, Incline Village, NV.

Damodaran, P., W. E. Wilhelm. 2004. Branch-and-price methods for prescribing profitable upgrades of high-technology products with stochastic demands. *Decision Sciences* **35** 55–82.

Dantzig, G. B., P. Wolfe. 1960. The decomposition principle for linear programs. *Operations Research* **8** 101–111.

Dash Associates, 1994. *XPRESS User manual.*

Day, P. R., D. M. Ryan. 1997. Flight attendant rostering for short-haul airline operations. *Operations Research* **45** 649–661.

Desrochers, M., F. Soumis. 1989. A column generation approach to the urban transit crew scheduling problem. *Transportation Science* **23** 1–13.

Desrosiers, J., H. B. Amor, F. Soumis, D. Villeneuve. 2002. Stabilized column generation. Workshop on Computational Methods for Large Scale Integer Programs, Institute for Mathematics and its Applications (IMA), University of Minnesota. http://www.ima.umn.edu/talks/workshops/10-14-19.2002/desrosiers/desrosiers.ppt (accessed July 2004).

Desrosiers J., Y. Dumas, M. M. Solomon, F. Soumis. 1995. Time constrained routing and scheduling. M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser eds. *Handbooks in Operations Research and Management Science, Volume 8: Network Routing*, Elsevier, Amsterdam, 35–140.

Du Merle, O., D. Villeneuve, J. Desrosiers, P. Hansen. 1999. Stabilized column generation. *Discrete Mathematics* **194** 229–237.

Ehrgott, M., D. M. Ryan. 2002. Constructing robust crew schedules with bicriteria optimization. *Journal of Multicriteria Decision Analysis* **11** 139–150.

Fisher, M. L., R. Jaikumar, L. N. Van Wassenhove. 1986. A multiplier adjustment method for the generalized assignment problem. *Management Science* **32** 1095–1103.

Ford, L. R., D. R. Fulkerson. 1958. A suggested computation for the maximal multicommodity network flows. *Management Science* **5** 97–101.

Gilmore, P. C., R. E. Gomory. 1961. A linear programming approach to the cutting stock problem. *Operations Research* **9** 849–859.

Girard A., Sansó, B. 1998. Multicommodity flow models, failure propagation, and reliable loss network design. *IEEE/ACM Transactions on Networking* **6** 82–93.

Guignard, M., M. B. Rosenwein. 1989. An improved dual based algorithm for the generalized assignment problem. *Operations Research* **37** 658–663.

IBM CORPORATION, 1990. Optimization Subroutine Library, Guide and Reference.

Johnson, E. L. 1989. Modeling and strong linear programs for mixed integer programming. S.W. Wallace eds. *Algorithms and Model Formulations in Mathematical Programming*, Springer-Verlag, 1–43.

Johnson, E. L., G. L. Nemhauser, M. W. P. Savelsbergh. 2000. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing* **12** 2–23.

Kallehauge, B., J. Larsen, O.B.G. Madsen. 2001. Lagrangean duality applied on vehicle routing with time windows. Technical Report IMM-TR-2001-9, Informatics and Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark.

Karabakal, N., J. C. Bean, J. R. Lohmann. 1992. A steepest descent multiplier adjustment method for the generalized assignment problem. Report 92-11, University of Michigan, Ann Arbor, MI. http://www-personal.engin.umich.edu/~nejat/academic/gap.zip

Kelley, J. E. 1960. The cutting-plane method for solving convex programs. *Journal of SIAM* **8** 703–712.

Kenyon, A. S., D. Morton. 2003. Stochastic vehicle routing with random travel times. *Transportation Science* **39** 69–82.

Kleywegt A. J., A. Shapiro, T. Homem-de-Mello. 2002. The Sample Average Approximation Method for Stochastic Discrete Optimization. *SIAM Journal on Optimization* **12** 479–502.

Ladanyi, L., J. J. Forrest, J. Kalagnanam. 2001. Column generation approach to the multiple knapsack problem with color constraints. Research Report RC-22013, IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

Laporte, G., F. V. Louveaux. 1993. The integer L-shaped method for stochastic integer programs with complete resource. *Operations Research Letters* **13** 133–142.

Laporte, G., F. V. Louveaux, L. Van Hamme. 1994. Exact solution to a location problem with stochastic demands. *Transportation Science* **28** 95–103.

Lasdon, L. S. 1970. *Optimization Theory for Large Systems*. Macmillan, New York.

Lin, S. 1965. Computer solutions of the traveling salesman problem. *Bell System Technical Journal* **44** 2245–2269

Lougee-Heimer, R. 2003. The Common Optimization INterface for Operations Research: promoting open-source software in the operations research community. *IBM Journal of Research and Development* **47** 57–66.

Louveaux F. V.,D. Peeters. 1992. A dual-based procedure for a stochastic facility location. *Operations Research* **40** 564–573.

Lübbecke, M. E., J. Desrosiers. 2002. Selected Topics in Column Generation. Les Cahiers de GERAD G-2002-64, Group for Research in Decision Analysis, Montreal, Canada. In http://www.optimization-online.org/DB_FILE/2002/12/580.pdf (accessed July 2004)

Lulli, G., S. Sen. 2004. A branch-and-price algorithm for multi-stage stochastic integer programming with application to stochastic batch-size problems. *Management Science*, accepted for publication.

Mak, W.-K., D. P. Morton, R. K. Wood. 1999. Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters* **24** 47–56.

Marsten, R.E. 1975. The use of boxstep method in discrete optimization. *Mathematical Programming Studies* **3** 127–144.

Marsten, R.E., W.W. Hogan, J.W. Blankenship. 1975. The boxstep method for large-scale optimization. *Operations Research* **23** 389–405.

Martello, S., D. Pisinger, P. Toth. 2000. New trends in exact algorithms for the 0-1 knapsack problem. *European Journal of Operational Research* **123** 325–332.

Martello, S., P. Toth. 1990. Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, New York.

Medard, C. P., N. Sawhney. 2004. Airline crew scheduling: From planning to operations. Carmen Research and Technology Report CRTR-0406. In http://www.carmensystems.com/research_development/articles/crtr0406.pdf (accessed July 2004)

MINTO in http://www.isye.gatech.edu/faculty/Martin_Savelsbergh/software (accessed July 2004).

Neame, P. 1999. Nonsmooth dual problems in integer programming. PhD Thesis, University of Melbourne, Australia.

Osman, I. H. 1995. Heuristics for the generalised assignment problem: Simulated annealing and tabu search approaches. *OR Spektrum* **17** 211–225.

Papagiannaki, K., S. Moon, C. Fraleigh, P. Thiran, C. Diot. 2003. Measurement and analysis of single-hop delay on an IP backbone network. *IEEE Journal on Selected Areas in Communications* **21** 908–921.

Ralphs, T. K., L. Ladanyi. 2001. *COIN/BCP User's Manual*. http://www-124.ibm.com/developerworks/opensource/coin/presentations/bcp-man.pdf.

Ralphs, T. K., L. Ladanyi, M. J. Saltzman. 2003. Parallel branch, cut, and price for large-scale discrete optimization. *Mathematical Programming* **98** 253–280.

Ribeiro C. C., F. Soumis. 1994. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research* **42** 41–52.

Rockafellar, R. T., R. J-B. Wets. 1976. Stochastic convex programming: relatively complete recourse and induced feasibility. *SIAM Journal on Control and Optimization* **14** 574–589.

Romeijn, H. E., D. H. Morales. 2001. Generating Experimental Data for the Generalized Assignment Problem. *Operations Research* **49** 866–878.

Ross, G. T., R. M. Soland. 1975. A branch-and-bound algorithm for the generalized assignment problem. *Mathematical Programming* **8** 91–103.

Ryan, D. M. 2004. Personal communication, 10 June.

Ryan, D. M., B.A. Foster. 1981. An integer programming approach to scheduling. A.Wren, ed. *Computer Scheduling of Public Transport, Urban Passenger Vehicle and Crew Scheduling*. North Holland, Amsterdam. 269–280.

Savelsbergh, M. W. P. 1997. A branch-and-price algorithm for the generalized assignment problem. *Operations Research* **45** 831–841.

Savelsbergh, M. W. P. 2001. Branch-and-price: Integer programming with column generation. C. Floudas, P. Pardalos, eds. *Encyclopedia of Optimization, Volume 1*, Kluwer Academic Publishers, Boston, MA. 218–221.

Savelsbergh, M. W. P., G. L. Nemhauser. 1998. Functional description of MINTO, a Mixed INTeger Optimizer (Version 3.0). Report COC-91-03D, Georgia Institute of Technology. http://www.isye.gatech.edu/faculty/Martin_Savelsbergh/software/doc30.pdf (accessed July 2004).

Savelsbergh, M. W. P., M. Sol 1998. DRIVE: Dynamic routing of independent vehicles. *Operations Research* **46** 474–490.

Sen, S., J. L. Higle. 2000. The C^3 theorem and a D^2 algorithm for large scale stochastic integer programming: set convexification. *Stochastic Programming E-Print Series*. (http://dochost.rz.hu-berlin.de/speps).

Shiina T., J. R. Birge. 2004. Stochastic unit commitment problem. *International Transactions in Operational Research* **11** 19–32.

Singh, K., A. Philpott, R. K. Wood. 2004. Column generation for designing survivable electric power networks. Working paper, Dept. of Engineering Science, University of Auckland, Auckland, New Zealand.

Spoerl, D., R. K. Wood. 2003. A stochastic generalized assignment problem. INFORMS Annual Meeting, Atlanta, GA, 19–22 Oct.

SYMPHONY. 2004. "Frequently Asked Questions" in http://branchandcut.org/ SYMPHONY/ faq.htm (accessed July 2004).

Vance P. H., C. Barnhart, E. L. Johnson, G. L. Nemhauser. 1997. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research* **45** 188–200.

Vanderbeck, F. 2000. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research* **48** 111–128.

Vanderbeck, F. 2003. A generic view at the Dantzig-Wolfe Decomposition approach in mixed integer programming. Working Paper U-02.22, Laboratoire de Mathématiques Appliquées Bordeaux (MAB), Université Bordeaux. http://www.math.u-bordeaux.fr/~fv/papers/dwcPap.pdf (accessed July 2004).

Vanderbeck, F., L. A. Wolsey. 1996. An exact algorithm for IP column generation. *Operations Research Letters* **19** 151–159.

Walkup, D. W., R. J.-B. Wets. 1967. Stochastic programs with recourse. *SIAM Journal on Applied Mathematics* **15** 1299–1314.

Weingartner, H. M., D. N. Ness. 1967. Methods for the multidimensional 0/1 knapsack problem. *Operations Research* **15** 83–103.

Wets, R.J.-B. 1966. Programming under uncertainty: the complete problem. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* **4** 316–339.

Wolsey, L. A. 1998. *Integer programming*. John Wiley & Sons, New York.

Zhou, J., B. Liu. 2003. New stochastic models for capacitated location-allocation problem. *Computers and Industrial Engineering* **45** 111–125.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center
Ft. Belvoir, Virginia

Dudley Knox Library
Naval Postgraduate School
Monterey, California